软件跟踪链自动化技术研究综述

汪 烨[1)] 胡 坤[1)] 姜 波[1)] 夏 鑫[2)] 唐贤书[1)]

[1)](浙江工商大学计算机科学与技术学院 杭州 310018)

[2)](华为技术有限公司软件工程应用技术实验室 杭州 310007)

摘 要 软件可跟踪性作为软件的一项重要能力,其目的是通过在不同的软件制品之间建立跟踪链,捕获、链接、追踪每一个重要的软件制品.近年来,将信息检索、自然语言处理、机器学习以及深度学习等技术用于软件跟踪链的创建、维护和验证,大大减少了开发人员手动处理跟踪链的成本,因此受到学术界和工业界的广泛关注.在本文中,我们着重从软件跟踪链的自动化创建、维护和验证等方面着手,对近十年来研究进展进行梳理和总结.主要内容包括:(1)统计并分析软件跟踪链创建、维护和验证的自动化方法和技术;(2)对软件跟踪链的应用研究进行总结;(3)汇总了当前软件跟踪链相关技术评估研究和工具支持;(4)从技术难点中归纳得出目前跟踪链相关自动化技术所存在的关键问题,围绕跟踪软件的复杂性、跟踪链的粒度问题、精度问题、类型受限问题、验证效率问题、应用规模和时间问题以及工具评估不全面问题这七个部分,阐述了上述问题的可能解决思路和未来发展趋势.

关键词 软件跟踪链;机器学习;人工智能;深度学习;自然语言处理

中图法分类号 TP18 **DOI号** 10.11897/SP.J.1016.2023.01919

A Systematic Literature Review of Software Traceability Links Automation Techniques

WANG Ye[1)] HU Kun[1)] JIANG Bo[1)] XIA Xin[2)] TANG Xian-Shu[1)]

[1)](*School of Computer Science and Technology, Zhejiang Gongshang University, Hangzhou* 310018)

[2)](*Software Engineering Application Technology Lab, Huawei, Hangzhou* 310007)

Abstract As an important software capability, software traceability aims to capture, link and trace each crucial software artifact via constructing the traceability links between them. The study of software traceability covers many aspects, including traceability modelling, traceability assessment and traceability implementation. Traceability links interconnect software artifacts with each other and use the resulting associative networks to resolve issues with software products and their development processes. Traceability links provide critical support for many software engineering activities, including impact analysis, software verification, test case selection, compliance verification, system security assurance and defect detection. Traceability links refer to a specific relationship between a pair of software artifacts, one of which is the source artifact and the other is the target artifact. They records various relationships between software artifacts such as dependencies, influences, and causal relationships. The direction of traceability links can be one-way or two-way. Various traceability links can help software developers to understand, develop

收稿日期:2022-05-31;在线发布日期:2023-01-18.本课题得到浙江省自然科学基金项目(LY21F020011, LY20F020027, LY19F020003)、电商可信交易关键技术研究及应用-基于跨境支付大数据的电商可信交易关键技术研究与应用(2021C01162)资助.

汪 烨,博士,副教授,硕士生导师,中国计算机学会(CCF)会员,主要研究方向为软件工程、机器学习和服务计算等.E-mail:yewang@zjgsu.edu.cn. **胡 坤**,硕士研究生,主要研究方向为软件工程、机器学习和服务计算等.**姜 波(通信作者)**,博士,教授,中国计算机学会(CCF)会员,主要研究方向为软件工程、机器学习和服务计算等.E-mail:nancybjiang@zjgsu.edu.cn. **夏 鑫**,博士,ARC DECRA研究员,主要研究方向为数据挖掘、软件工程和机器学习等. **唐贤书**,本科生,主要研究方向为软件工程、机器学习等.

and manage systems both efficiently and effectively. At the same time, traceability links can help people involved in all phases of software development activities to accomplish their development tasks. Requirements traceability links, as the most widely used traceability links, enable the construction and maintenance of traceability links between requirements and other software artifacts. Moreover, traceability links also include the establishment of links between code and tests, design and code, models and code, defects and code, and so on. In recent years, the creation, maintenance and validation of traceability links with information retrieval, natural language processing, machine learning, and deep learning can reduce the manual handling cost of traceability links by developers, and therefore have received extensive attentions from academia and industry. There are also some works reviewing software traceability links approaches and techniques. In this paper, we focus on the automation techniques of the creation, maintenance and validation of traceability links so as to sort out and summarize the research progress in the past ten years. The main content includes the statistics and analysis of approaches and techniques for automated creation, maintenance and validation of traceability links, the application research of traceability links, the state-of-the-art traceability links related evaluation research and tools support, and the key problems of the current traceability links techniques. The problems are summarized from the technical difficulties around seventh parts: the complexity of the tracing software, the granularity problem, the unsatisfying accuracy, the type limitation, the validation efficiency, the application scale and time, and the incomprehensive evaluation of traceability links.. Besides, several possible solution ideas and future development trends of the problems are elaborated, including the construction of horizontal traceability links between software artifacts, the scalable and configurable automation techniques of traceability links, the integration of traditional approaches with artificial intelligence techniques, the creation of multiple types of traceability links using intermediary artifacts, the interactive verification of traceability links, the real-time retrieval of traceability links and open sourcing of related source codes. This review also reveals that: (1) The creation of traceability links has received a lot of academic attentions, but the research on the maintenance, verification, and application of traceability links needs more attention; (2) Requirements-to-code links are the most concerned type of researchers, followed by requirements-to-design and design-to-code, while other traceability links such as code/data-to-model and screenshot-to-defect are also starting to enter the vision of researchers; (3) With the continuous development of artificial intelligence (AI), AI-based techniques such as Naive Bayes, SVM, Bert, Doc2Vec, RNN have been widely used in the creation, maintenance and verification of traceability links; (4) In the creation of traceability links, it is difficult to achieve good results by relying only on information retrieval and artificial intelligence techniques. Information retrieval, machine learning or deep learning techniques should be combined with traditional heuristics, model-based methods and so on, to make up for the deficiencies in AI technologies and traditional methods to further improve the quality of traceability links; (5) Research on traceability links automation techniques in complex environments, cross-platform and cross-language should be on the agenda in the future.

Keywords software traceability links; machine learning; artificial intelligence; deep learning; natural language processing

1 引 言

软件可跟踪性(Software Traceability)是将软件制品与其他制品相互关联、维护关联链接,并使用所产生的关联网络来解决软件产品及其开发过程中问题的能力[1].可跟踪性为众多软件工程活动提供了关键的支持,包括软件验证[2]、测试用例选择[3]、

合规性验证[4]、缺陷定位[5]、系统安全性保障[6]和变更影响分析[7].

软件可跟踪性的研究涵盖很多方面,包括可跟踪性建模、可跟踪性评估和可跟踪性实施等[8].在项目实践中,软件跟踪链(Traceability Links/Trace Links/Traces)作为可跟踪性的实施基础,成为当前软件工程的研究热点之一[9].软件跟踪链[10-11](以下简称跟踪链)是指一对软件制品之间的单向或双向特定关联,记录了源制品与目标制品之间的各种依赖性、影响、因果关系等.需求跟踪链作为软件跟踪领域应用最广泛的一种,实现了需求和其他软件制品之间跟踪链的构建和维护.此外,软件跟踪链还包括建立代码与测试、设计与代码、模型与代码、缺陷与代码等之间的关联.

研究表明,处理大规模的复杂系统时,构建和维护跟踪链是一项劳动密集、艰巨且容易出错的任务.研究者们提出了很多跟踪链创建、维护和验证技术,包括编辑距离[7]、滑动窗口[12-13]、正则表达式(Regular Expression,RE)[9]、基于模型的方法[14-16]、关键短语(Key Phrases,KP)[9,17]等.但上述传统方法存在以下不足:(1)需要手动或半自动建立制品之间的链接,这一过程需要消耗大量的时间和精力;(2)制品之间建立跟踪链的准确性、完整性和一致性较低.之后国内外提出了诸多将机器学习[18]、深度学习[19]、自然语言处理[20]等自动化技术,以更高的精度和覆盖范围,自动地在各种类型的软件制品之间建立跟踪链[5,12-14,21-24],这些方法极大提高了跟踪链的精确性、一致性和完整性,且大幅度减少了手工操作,降低了人工操作的成本.

目前国内外已对跟踪链的相关研究进行归纳总结.早在 2008 年,Torkar 等人[25]就统计了 1997 到 2007 年的需求跟踪定义、方法、工具和技术,并总结了相应的工业界案例.2009 年,Winkler 等人[10]从需求工程和模型驱动工程两个领域总结软件跟踪的研究进展,分析了两个领域研究的共性、差别和挑战.2013 年,Borg 等人[26-27]调研了信息检索技术在跟踪链构建中的应用.2014 年,Javed 等人[28]和 Parizi 等人[29]分别调研了软件架构和代码之间、测试与代码之间的跟踪链相关技术.Cleland-Huang 等人[30]从面向目标、面向过程、技术-基础设施三个视角对 2004 年至 2014 年的软件跟踪研究进行系统地分析整理,从可跟踪性的计划与管理、跟踪链创建和跟踪链的使用三个方面提出未来软件跟踪的研究方向.胡成海等人[31]针对基于信息检索的需求跟踪方法,从技术改进、支撑工具和度量指标进行了充分的调研.Wang 等人[32]梳理了 2006 年到 2016 年的需求跟踪工作,着重对需求跟踪的挑战和技术进行分析总结,为开发人员在实际项目中选择合适的技术提供决策支持.Aung 等人在 2020 年[33]调研了跟踪链创建对变更影响分析的作用,包括跟踪链创建技术、变更影响分析集合、评估指标、跟踪链方向等方面.Tian 等人[34]在 2021 年采用映射研究方法分析了软件跟踪对系统演化和维护的影响.表 1 列出了本文工作与上述综述的区别.为了使研究者对跟踪链自动化技术有一个更清晰、更全面的把握,本文采用文献综述的方法,对近十年的研究进行了系统地分析整理,并对新方法和新技术进行了统一的调研.

软件跟踪链自动化技术的研究集中于以下五个研究问题[8,30-35]:

(1)跟踪链创建[33]:通过自然语言处理、信息检索、深度学习、机器学习等技术,在软件的源制品和目标制品之间建立跟踪链,用于维护制品间的关系.

(2)跟踪链维护[34]:对已创建的跟踪链进行维护的过程.

(3)跟踪链验证[30,35]:对软件跟踪链的正确性、一致性和完整性进行验证或评估.

(4)跟踪链应用[36]:将跟踪链应用于软件开发活动中,如需求评估和管理、程序理解、软件维护等,以更直观的方法展示软件跟踪链的价值.

(5)评估研究[31]:对现有跟踪链相关自动化技术进行实验评估.

如图 1 所示,跟踪链的创建、维护、验证以及应用属于新方法和新技术,本文第三、四节将对这四个研究问题及其相关工作分别进行总结,第五节介绍了相关自动化方法和技术的评估研究,包括数据集和评估指标,第六节总结近十年文献中所使用的跟踪链自动化工具,第七节阐述目前跟踪链在创建、维护、验证、应用、评估和工具方面所存在的关键问题和解决方案.图 2 展示了软件跟踪链自动化研究的一般过程框架.

2　文献筛选

在文献选取上,我们参考了常用系统文献综述方法[37-38],选取的文献遵循了以下标准:(1)该文献针对跟踪链的自动化创建、维护、验证、应用等方面提出了新技术,且取得了一定的效果;(2)该文献对

现有跟踪链自动化技术进行改进和评估;(3)该文献已经公开发表在国内外的专著、会议、期刊中,并且具有一定的影响力.

根据三个标准,本文通过四个步骤筛选文献.

表 1 相关综述比较

综述	关注问题	时间范围	制品范围	文献数量	文献来源
Torkar 等人[25]	(1)需求跟踪的定义、方法、工具、技术;(2)需求跟踪的挑战工业界调研	1997～2007	需求与其他制品	52	IEEE/ACM/Springer/Inspec/EI
Winkler 等人[10]	(1)需求工程和模型驱动工程中两个领域中跟踪研究的共性、差别和挑战;(2)跟踪计划、跟踪记录、跟踪使用和跟踪维护	2010 年前	多种制品	—	—
Borg 等人[26-27]	跟踪链创建中的信息检索技术及精度比较	2003～2013	多种制品	79	IEEE/ACM/Web of science/Inspec/EI/SciVerse
Javed 等人[28]	软件架构和代码之间的跟踪方法、类型、粒度、方向、展示和挑战	1999～2013	架构与代码	11	ACM
Parizi 等人[29]	测试与代码之间的跟踪链创建技术的自动化程度、工具支持、类型、可视化、评估、扩展性和挑战	2014 年前	测试与代码	—	—
Cleland-Huang 等人[30]	(1)跟踪链的计划与管理、跟踪链创建、维护和使用技术;(2)跟踪链的目标;成本效益、基于价值、可配置、可扩展性等	2003～2013	多种制品	72	TSE/ASE/ICSE/RE/ASE/Software/ICSM
胡成海等人[31]	基于信息检索的需求跟踪技术、改进策略、支撑工具和度量指标	2004～2014	需求与其他制品	—	—
Wang 等人[32]	(1)需求跟踪的挑战:自动、可信、可扩展、协作性、动态化、非功能需求跟踪、轻量化、展示、价值和成本效益; (2)技术:需求跟踪链生成、展示、维护、应用和非功能需求跟踪技术	2006～2016	需求与其他制品	114	IEEE/ACM/Springer/Elsevier/EI
Aung 等人[33]	跟踪链创建技术、方向、工具支持、数据集和影响的变更集合	2012～2019	多种制品	33	ACM/IEEE/ Science Direct/SpringerLink/Scopus
Tian 等人[34]	与软件演化和维护相关的跟踪技术和工具	2000～2020	多种制品	63	ACM/IEEE/Springer/Science/Direct/Wiley/InterScience/EI/Web of Science
本文	(1)跟踪链创建、维护、验证和应用的自动化技术; (2)软件跟踪链的精度、类型、数据集、工具和评估指标	2011～2021	多种制品	117	《中国计算机协会推荐国际会议和期刊目录》B 类以及 B 类以上的权威软件工程期刊和会议;《计算领域高质量科技期刊分级目录》T1 类科技期刊

(1)根据研究方向确定检索的关键词,确定的英文关键词为:Software Tracing、Software Traceability、Requirements Tracing、Requirements Traceability、Traceability Links、Trace Links、Traceability Recovery、Trace Retrieval、Trace Creation、Trace Generation、Trace Validation、Trace Maintenance 等,确定的中文关键词包括:软件跟踪、软件追踪、需求跟踪、需求追踪、跟踪链恢复、跟踪链创建、跟踪链生成、跟踪链维护、跟踪链验证等.主要的数据库包括:IEEE Xplore Digital,Springer Link Online Library,ACM Digital Library,ScienceDirect Library,ISI Web of Science, Engineering Village 和中国知网.检索时间区间定义在 2011 年 1 月～2021 年 12 月.

(2)按照标题、关键词、摘要、结论和来源的优先顺序对上述的文献进行筛选,其标准为:①根据《中国计算机协会推荐国际会议和期刊目录》,选择 B 类以及 B 类以上的权威软件工程英文期刊和会议;②根据《计算领域高质量科技期刊分级目录》选择 T1 类的中文科技期刊;③与跟踪链自动化技术有关.由此得到了 132 篇文献.

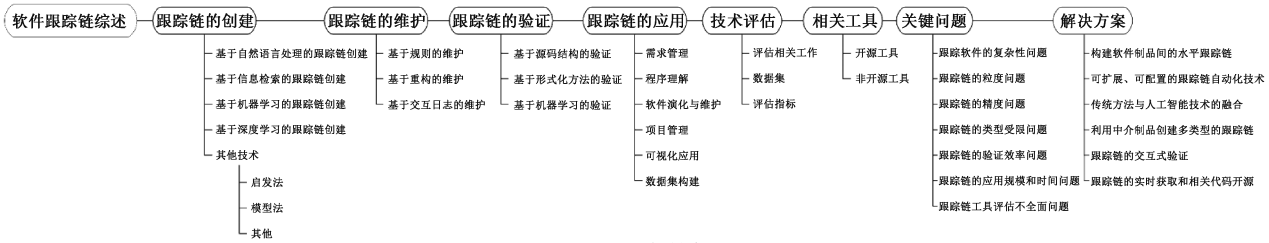

图 1 文章结构

(3)由第一作者和第二作者对 132 篇文献进行全文排查.对于同一篇文献,如果这两名研究人员持

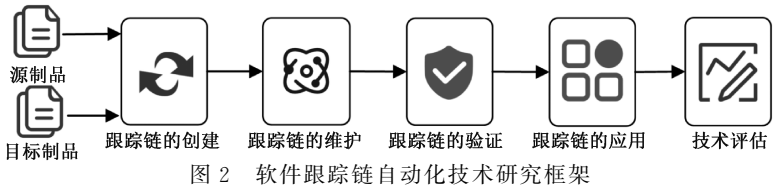

图 2　软件跟踪链自动化技术研究框架

不同意见时,则由其他人员参与判断.经过上述步骤,保留相关文献 110 篇.

(4)得到步骤 3 的相关文献之后,我们再从这些参考文献中通过滚雪球的方式[39]根据筛选的标准进行筛选.

通过上述步骤对文献进行筛选之后,共得到 117 篇相关文献,这些文献都是与跟踪链自动化研究直接相关,对跟踪链创建、维护、验证和应用以及跟踪链的评估研究提供了新思路.图 3、图 4 分别展示了本文所总结文献发表年份和来源的分布情况.

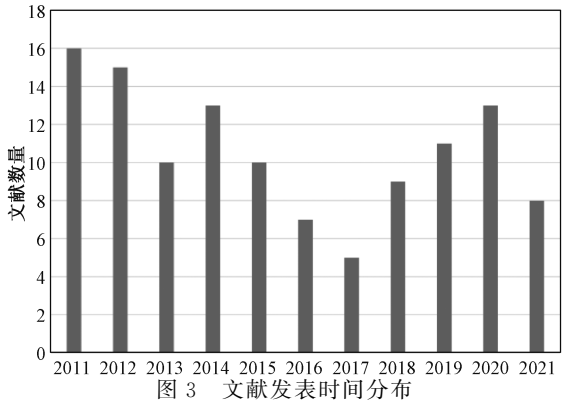

图 3　文献发表时间分布

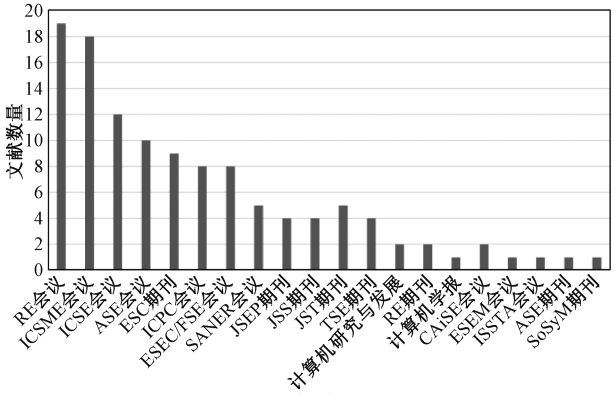

图 4　文献来源分布

3　软件跟踪链的创建、维护与验证

本文将跟踪链按照制品不同分为以下类别:需求-需求(R-R)、需求-代码(R-C)、需求-设计(R-D)、需求-测试(R-T)、需求-模型(R-M)、模型-代码(M-C)、设计-测试(D-T)、设计-代码(D-C)、代码-测试(C-T)、代码-缺陷(C-B)、代码/数据-模型(C/Dt-M)、代码-文档(C-Doc)、代码提交-问题(CC-I)、文档-文档(Doc-Doc)、代码-代码(类-类,CL-CL)、代码提交-缺陷(CC-B)、屏幕截图-缺陷(SS-B)和代码提交-代码(CC-C).上述类别与本文所选择的文献范围有关,对于未明确说明跟踪链类别的文献,本文以其采用的数据集类型为准.图 5 展示了各个跟踪链类型的文献数量.

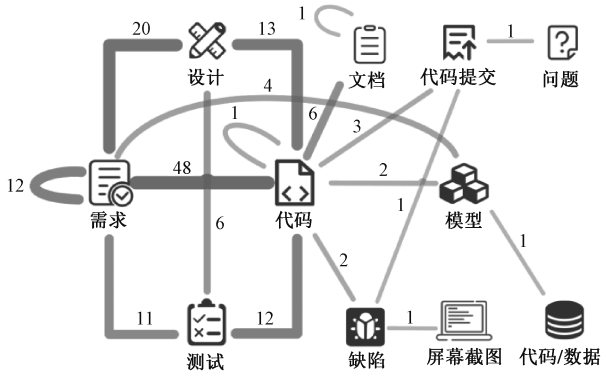

图 5　跟踪链类型统计

3.1　跟踪链的创建

关于跟踪链的创建,国内外许多研究者们已提出大量优化和改进的新技术.本文将其分为基于自然语言处理(Natural Language Processing, NLP)、基于信息检索(Information Retrieval, IR)、基于机器学习(Machine Learning, ML)、基于深度学习(Deep Learning, DL)的技术、启发法、模型法和其他技术.图 6 总结了各种技术所占比例,图 7 总结了 2011～2021 年各种技术的发展趋势.近十年来,ML 和 IR 技术为跟踪链创建所采用的主流技术,在 2011～2013 年以 IR 技术为主,其次为启发法和模型法,从 2017 年开始 ML 成为主流技术,DL 技术也被越来越多的学者使用.

3.1.1　基于 NLP 的跟踪链创建

基于 NLP 的方法通常对软件制品中采用自然语言描述的文档进行挖掘和分析,计算其语法或语义逻辑,通过关键词或信息匹配实现跟踪链的创建,包括基于句法分析、基于谓词逻辑和基于语义模型的方法.王金水等人[40]提出一种基于句法分析的动态需求跟踪方法,该方法利用语句分析技术从制品中抽取出最有可能刻画自身特征的标引词,再通过

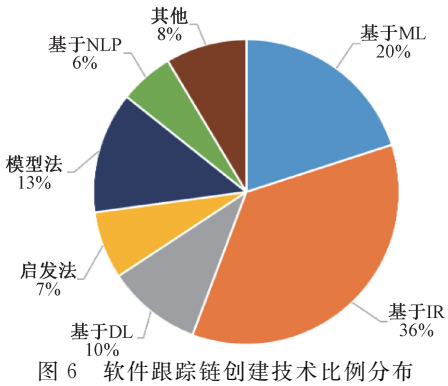

图 6 软件跟踪链创建技术比例分布

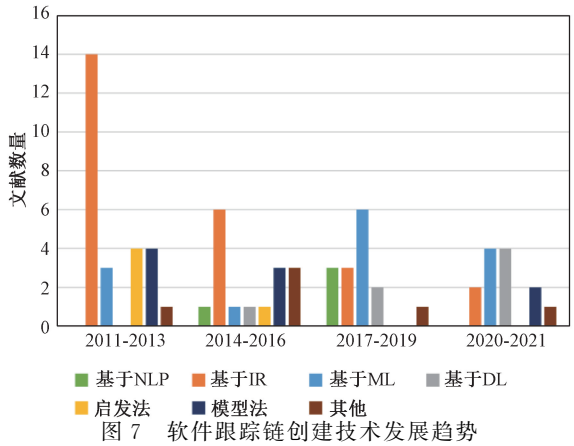

图 7 软件跟踪链创建技术发展趋势

语义聚类创建不同制品之间的跟踪链. 该方法可减少制品中噪音对需求跟踪带来的不利影响. 鉴于当前缺乏需求与设计间跟踪链的研究,王飞等人[41]提出了一种基于谓词逻辑的需求到设计的跟踪链创建方法:首先,定义一个基于谓词逻辑的跟踪信息,并分别定义需求横向与纵向跟踪关系,然后再基于语义模型给出跟踪关系的自动推导和检验规则,以建立精确、完整的跟踪链. 为了创建需求之间的语义链接,Schlutter 等人[42]提出了一种语义关系图和扩散激活(Spreading Activation, AS)的方法. 该方法采用 NLP 管道将自然语言需求自动转化成语义关系图,使术语和其关系转化成顶点和边,利用扩散激活作为语义搜索算法,对给定的查询查找其所有相关信息,然后通过结构或语义找到相关链接. 与之类似的还有 Lapeña 等[43]提出的本体模型扩展方法.

表 2 总结了基于 NLP 的跟踪链创建研究. 上述单独采用传统 NLP 中的语法、语义分析的技术,跟踪链的精度并不理想. 目前越来越多的跟踪链创建是将 NLP 与其他信息检索、机器学习、深度学习技术相结合,从而提高跟踪链识别的准确性和效率.

3.1.2 基于 IR 的跟踪链创建

信息检索 (Information Retrieval, IR) 作为一种应用在跟踪链创建的自动化技术已获得广泛接受. 基于 IR 的方法通过计算不同制品间的文本相似性来构建软件制品之间的链接,即相似度越高,两个制品之间存在跟踪链的可能性就越大[44,60]. 现有基于 IR 的跟踪链创建研究都是在已有 IR 技术上进行优化和增强,以解决常见问题,如从候选跟踪链列表中过滤噪声,消除不同制品之间词义不匹配问题等,从而提高跟踪链精度. 向量空间模型(Vector Space Model, VSM)、潜在语义索引(Latent Semantic Indexing, LSI)和 JS 散度模型(Jensen and Shannon Models, JSMs)是目前 IR 中最常用的三种模型. 表 3 统计了基于 IR 的跟踪链创建研究,由于该类技术较多,在图 8 中给出了这些新技术的演变过程.

表 2 基于 NLP 的跟踪链创建研究

方法/技术	跟踪链	文献
句法分析	R-R、R-C、R-T、CL-CL、C-T	[40]
谓词逻辑	R-D	[41]
语义关系图和扩散激活	R-R	[42]
本体模型扩展	R-M	[43]

根据 IR 技术与其他方法融合的阶段不同,我们将基于 IR 的跟踪链创建分为以下三类:

(1)优化数据预处理. 将需要建立跟踪链的源制品和目标制品输入到 IR 引擎之前,对它们进行文本预处理以去除噪声信息,在预处理的过程中融合了其他方法进行优化;

(2)改进相似度计算. 优化相似度计算方法,计算两个制品的相似度以得到候选跟踪链;

(3)精化链接. 经过相似度计算后,通过自动或半自动的方式对候选跟踪链进行精化.

在数据预处理阶段,可通过正则表达式[9]、平滑过滤器[45-46]、术语名词[47]、ConPOS[48]、扩展语料库[49]、学习规则[50]、重构[51]、动态切片和概念耦合[52]等技术来优化已有的 IR 技术. 例如,De Lucia 等[45-46]利用平滑过滤器改进原有的 VSM、LSI 和 JS 技术来创建代码与需求、测试以及其他文档之间的链接. 又如,Capobianco 等人[47]提出了一种基于 IR 的启发式方法,该方法通过更专注于软件文档中出现的术语名词提高了精确率. 然而,该方法只考虑了名词,并未考虑到动词,导致重要语义信息丢失,Ali 等人[48]开发的 ConPOS 方法则很好地解决了这个问题. ConPOS 从需求文档中提取所有的 POS 类别(包括名词、动词、形容词、副词和代词),使用 VSM 和 JS 模型构建给定需求和代码之间的跟踪链,再采用剪枝策略优化跟踪链. Dasgupta 等[49]则提出一种

表 3　基于 IR 的跟踪链创建研究

方法/技术	跟踪链	增强阶段			文献
		P	S	R	
VSM、正则表达式、关键短语、聚类	C-Doc	+	+	+	文献[9]
VSM、代码所有权	R-C			+	文献[44]
VSM/LSI/JS、平滑过滤器	R-C、D-C、C-T	+			文献[45-46]
VSM/LSI/JS、基于名词的检索	R-C、R-R、R-T、M-C、C-T	+			文献[47]
VSM/JS、ConPOS	R-C	+			文献[48]
VSM/JS、扩展语料库	R-C	+			文献[49]
VSM、学习规则	CC-B	+			文献[50]
VSM、重构	R-C	+			文献[51]
LSI、动态切片、概念耦合	C-T	+			文献[52]
VSM、TF-IDF	R-R		+		文献[53]
VSM/JS、关系主题建模	R-C、D-C、C-T		+		文献[54]
VSM、跟踪查询转换	Doc-Doc		+		文献[55]
显式语义分析	R-C、R-R		+		文献[56]
VSM、CLM	R-C、R-D、R-T、D-C、D-T、C-T		+		文献[57]
VSM/LSI、CLM	R-C、R-D、D-C、R-T、C-T、D-T		+		文献[58]
VSM、多抽象关注点	C-B		+		文献[59]
VSM/JS、信息融合	R-C		+		文献[60-61]
VSM、DoCIT	R-D		+		文献[62]
VSM、紧密度分析	R-C			+	文献[63-64]
VSM、紧密度分析、用户反馈	R-C			+	文献[65]
VSM、基于日志和用户反馈	R-C			+	文献[66]
VSM、启发式	R-C			+	文献[67]

注:P:数据预处理阶段,S:相似度计算阶段,R:链接精化阶段.

使用外部函数调用文档和相关词集获得的相关文档来扩展语料库,从而进一步提高跟踪链的准确性. Wu 等人[50]开发的 ReLink 方法通过学习规则来优化数据预处理阶段,用来创建变更日志到缺陷之间的跟踪链. Mahmoud 等人[51]基于重构方法重新建立系统的扭曲词汇轨迹,从而创建需求到代码之间的跟踪链. Abdallah Qusef 等[52]采用动态切片、概念耦合与 LSI 相结合的方法创建测试和代码之间的跟踪链.

在相似度计算阶段,可采用关键短语[9]、TF-IDF[53]、关系主题建模[54]、跟踪查询转换[55]、显式语义分析[56]、连接链接方法(Connecting Links Method,CLM)[57-58]、多抽象关注点[59]和信息融合[60-61]对现有的 IR 技术进行优化. Heck 等[53]使用 TF-IDF 和 VSM 来计算需求文本相似度,在水平方向上建立 JIT(Just-in-Time)需求之间的跟踪链. 对于需求与其他制品之间的跟踪,Gethers 等[54]尝试将关系主题模型与 VSM 和 JS 模型相结合. Dietrich 等人[55]则利用跟踪查询转换(Trace Query Transformation,TQT)技术,将用户反馈的好处扩展到多个跟踪查询中,通过一种新形式的关联规则挖掘查询转换规则,提高跟踪查询的效率和跟踪链的质量. 为了解决专家只能在后期干预软件跟踪这一问题,Mahmoud 等人[56]提出一种语义相关性(Semantic Relatedness)方法,通过将专家决策集成到底层检索机制,使专家决策能够尽早地参与到跟踪的早期阶段. 同时,该方法使用维基百科的语义关联度量,即显式语义分析(Explicit Semantic Analysis,ESA)作为基本检索机制. 结果显示,这种方法在召回率上优于 VSM 方法,而精确率上优于 LSI 方法,且在不同的阈值级别显示出更稳定的性能. 为了扩大跟踪链的通用性,使创建方法不仅局限于单一的跟踪链,Nishikawa 等人[57]提出一种基于 VSM 的 CLM 方法,用以生成两个通用制品之间的跟踪链,其核心思想就是使用第三个制品来构建两个制品之间的可传递跟踪链,但如何确定合适的跟踪链数量是该方法的关键. 由于 CLM 只使用单个数据集评估其方法,Alberto 等[58]基于 CLM,在多个数据集上复现了 Nishikawa 等人的实验,并利用聚合思想优化了该方法. 为了更加快速地生成关注点(如缺陷报告、软件特征请求)和代码方法之间的跟踪链,Zhang[59]等人提出了一种名为 MULAB 的多抽象关注点定位技术,通过考虑所有抽象级别来实现代码单元和关注点之间的相似性,然后结合 VSM 和多个模型计算相似度,并采用遗传算法推断最优主题模型的配置. 未来该方法可与文本挖掘如释义检测、深度学习等技术相结合来提高性能. Guo 等人[62]将 VSM 与一种领域情境化智能跟踪的解决方案 DoCIT 相结合,使链接创建精确率提高约 26%.

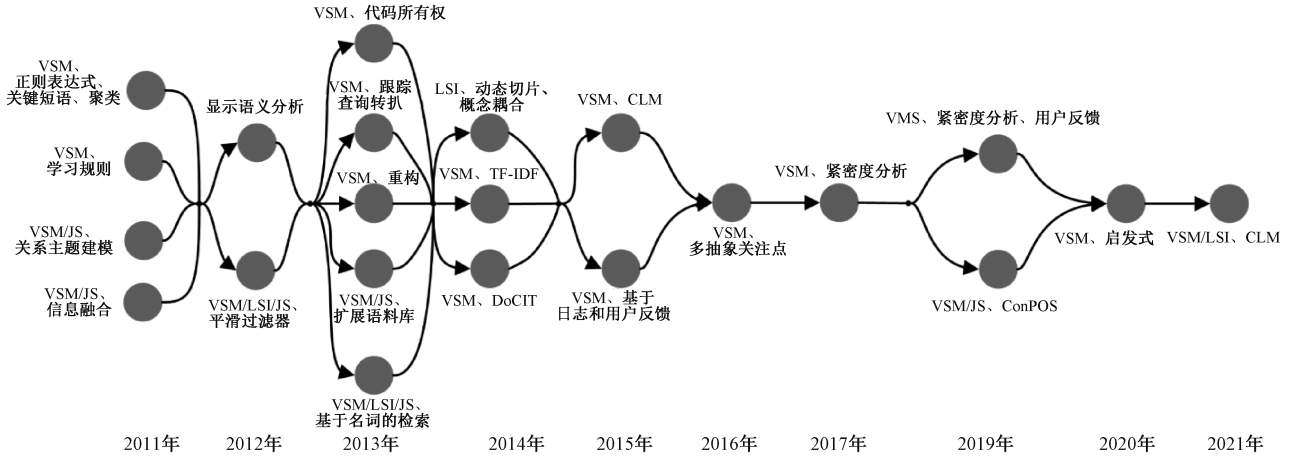

图 8 基于 IR 的跟踪链创建技术演变过程

在链接精化阶段，可结合聚类[9]、代码所有权[44]、代码依赖关系[63-64]、用户反馈[65-66]和启发式规则[67]等方法来进行 IR 技术的效果增强. 对于文档与代码之间的跟踪，陈小凡等人[9]提出一种将正则表达式、关键短语和聚类与 IR 结合的方法，其中聚类方法用于在链接精化阶段，以此克服传统 VSM 技术的局限性. 结果表明，相比单独使用 IR 技术，组合技术在创建精确率上大约提高了 5%. Diaz 等人[44]提出一种利用代码所有权(Code Ownership)的信息检索方法，先提取每个组件的作者，为每个作者确定其工作的“上下文”，计算需求与上下文之间的相似性并进行排名，再计算需求和代码之间的相似性，得到候选链接表. 最后根据上下文相似性排名，对候选链接表中的链接进行重新排名. 但该方法只考虑每个组件的代码所有权只属于单个作者，并没有考虑多个作者共同拥有代码所有权的情况. Kuang 等人[63]讨论了方法调用关系和数据依赖关系对于创建需求和代码之间跟踪链的影响，随后他们[64]又提出一种将 VSM 与紧密度(Closeness)分析相结合的技术，通过量化方法调用的紧密度和类间的数据依赖来提高跟踪链候选列表的排名. 通过三个真实案例系统的评估，证明了该方法的精确率优于三种基线方法. Kuang 等人[65]后续又提出一种基于用户反馈的方法，此方法结合代码紧密度分析与用户反馈来改进基于 IR 的方法. 首先定位一组候选区域，验证该区域中排名靠前的类与给定需求的相关性，再将经过验证的链接作为输入来重新为未验证的候选链接排序. Tsuchiya 等人[66]采用两种技术对 VSM 方法进行改进：(1)基于日志的跟踪方法；(2)基于用户反馈和调用关系的链接推荐. 然而目前链接推荐方法只针对代码中的调用关系，还需对其他代码关系进行验证.

从图 8 中，可以看出 VSM 是跟踪链创建中使用较多的 IR 技术，其次是 LSI 和 JS. 当前基于 IR 的跟踪链创建技术通过在三个不同阶段与其它技术结合，如代码所有权、紧密度、语义分析等，优化了传统方法效率的同时，也提高了跟踪链质量.

3.1.3 基于 ML 的跟踪链创建

跟踪链创建所采用的机器学习算法主要包括决策树(Decision Tree，DT)、KNN(K-Nearest Neighbor)、SVM(Support Vector Machine)[68]、朴素贝叶斯[69]等. 基于 ML 的方法侧重于优化制品间跟踪链的生成时间和人力成本.

采用机器学习方法创建跟踪链通常包括四个阶段：(1)文本处理：对文本进行分词、词序标注等操作，消除文本中含有的噪音，减少文本的冗余度；(2)文本特征提取：利用 Word2Vec、TF-IDF 等相关技术，将文本中的特征进行提取，使其能够输入模型中进行处理；(3)学习模型的构建和训练：首先根据任务的特点选择合适的机器学习模型，然后利用构建好的训练集进行训练，并使用验证集作为标准，不断调整参数，最优化模型；(4)跟踪链生成：将测试集的数据输入到训练好的模型中，根据模型的结果生成跟踪链. 表 4 统计了基于 ML 的跟踪链创建研究，并在图 9 中给出了这些新技术的演变过程.

为了提高软件特征定位技术的准确性，Abukwaik 等人[68]提出了一种功能注释推荐方法，能够帮助开发人员在新添加的代码中添加注释，通过比较 KNN、DT、SVM 这三种机器学习的算法，最后得出 KNN 表现最佳. 虽然该方法可以有效地构建跟踪链，但是需要足够的标签数据来训练分类器. 因此 Mills 等[70]又提出一种基于主动学习的方法(Active

Learning for ClAssification-based TRAceability Links, ALCATRAL),大大减少了有监督分类在构建跟踪链时所需的训练数据量,同时不影响算法性能.

表 4　基于 ML 的跟踪链创建研究

方法/技术	跟踪链	文献
层次贝叶斯网络	R-R、R-T	[6]
自定义分类器	D-C	[18]
注释推荐、KNN/DT/SVM	CC-C	[68]
流程和文本相关特征的组合、朴素贝叶斯/决策树/随机森林/主动学习	CC-I	[69]
	R-R、R-C、C-T、D-C、D-T、R-D	[70]
LSA/基于词库的方法/共现方法、分区聚类/层次聚类法	R-C	[71]
动态跟踪配置	R-C、R-D、R-T	[72]
进化计算、学习排序	R-M	[73]
机器学习分类器、TIM	R-M	[74]
层次聚类、K-means、二分聚类	R-C、R-D	[75]
K-means	R-R	[76]
ADTree/Bagging/FLR/IBK/LogitBoost/朴素贝叶斯/ZeroR	R-R、R-C、M-C、C-T、R-D、D-T、R-T	[77]
机器学习分类器	R-R、R-C、C-T、D-C、D-T、R-D	[78]
MSR、静态代码分析	C/Dt-M	[79]

为了实现代码与非功能需求之间的跟踪链,Mahmoud 等人[71]提出了一种基于无监督学习的非功能需求检查、分类和跟踪的方法.该方法利用非功能需求文本的语义来推断系统中潜在的质量约束,系统地对比了 LSA(Latent Semantic Analysis)、SVM、基于词表、分区聚类和层次聚类的方法,通过计算非功能需求与代码的文本相似度,创建它们之间的跟踪链.由于手动建立代码与架构设计之间的跟踪链耗费较大成本,Mirakhorli 等人[18]提出一种自动创建代码和设计之间跟踪链的新方法,该方法将策略跟踪信息模型(tactic Traceability Information Models,tTIMs)的概念与现有的跟踪检索和分类概念相结合,利用机器学习和轻量级结构分析出与设计策略相关的类别.然而,该方法中设计策略的正确性和完整性过于依赖研究人员主观判断.

为了解决代码提交与问题之间的链接不完整问题,Rath 等人[69]结合表征问题、代码更改的流程以及与文本相关的特征来训练朴素贝叶斯、J48 决策树和随机森林的分类器,从而识别提交记录中缺失的问题标签,并且生成缺失的链接.该方法能够以平均 96%的召回率和 33%的精确率有效地创建用于标记问题的跟踪链.对于不同制品之间的跟踪链,Moran 等人[6]设计并实现了分层概率模型(HierarchiCal PrObabilistic Model for SoftwarE Traceability, COMET).该模型使用了多个优先级,形成层次贝叶斯网络(Hierarchical Bayesian Network, HBN),再结合多个文本相似性度量的信息,得出两个给定制品之间存在链接的概率,其平均精度提高了约 5%.但是该方法的性能还需结合其它增强策略如用户反馈分析、平滑过滤器和查询扩展等来提高.

为了构建统一的多种跟踪链创建机制,Lohar 等人[72]提出了动态跟踪配置(Dynamic Tracking Configuration, DTC)的方法创建需求、测试、代码和设计之间的跟踪链.该方法将源制品和目标制品的训练集以及经过验证的跟踪链矩阵作为输入,以这些链接作为"参考集"来提高生成的跟踪链的准确性.

机器学习技术还被应用于需求到模型的软件跟踪链创建中.例如,Marcén 等[73]提出一种名为 TLR-ELtoR(Evolutionary Learning to Rank for Traceability Link Recovery)的新方法,结合了进化学习算法和机器学习技术.为了构建需求中的问题描述到领域模型的跟踪链,Saini[74]等人提出了一种基于机器学习的领域模型决策跟踪方法,该方法首先构建了一个跟踪信息模型(Traceability Information Model, TIM),然后通过描述层、预测层、合约层和跟踪层四层机制将不同级别模型元素的跟踪图进行集成,最后形成统一的跟踪知识图谱.

为了提高跟踪链的质量,Niu[75]等人提出一种基于聚类的方法,研究了 5 种聚类方法(K-means、二分法、3 种凝聚层次聚类变体)以及 3 种启发式算法(MAX、AVE、MED). K-means 聚类算法还被 Mezghani 等人[76]用在需求之间的跟踪链生成上. Falessi 等人[77]提出了一种剩余链接数评估方法,用以判断跟踪链列表中还剩多少链接未被识别.他们训练了 7 个 ML 分类器 ADTree、Bagging、FLR、

IBK、LogitBoost、ZeroR 和朴素贝叶斯，从列表中识别正确和错误的链接，结果发现 ADTree 的精确率最高，ZeroR 最低. Mills 等[78]通过设计一个二分类任务实现跟踪链创建的自动化. 他们利用机器学习分类器(包括 J48、KNN、朴素贝叶斯和随机森林)自动将所有候选链接推荐列表中的每个链接分类为有效或无效.

为构建机器学习仓库和软件构件之间的链接，Njomou 等人[79]结合软件资源库挖掘(Mining Software Repositories，MSR)和静态代码分析(Static Code Analysis，SCA)技术，识别机器学习模型、代码和数据制品，重构它们之间的链接.

由图 9 可知，使用最多的 ML 技术是朴素贝叶斯、层次聚类、LSA，此外，很多研究人员偏好使用自定义的机器学习分类器. 与 IR 技术相比，ML 技术创建的跟踪链在精确率和召回率的平衡上表现更好. 如果采用的是半监督或无监督学习方法，将大大减少需要标记的数据，进一步提高学习效率. 但是，基于 ML 的跟踪链创建仍存在一些问题，例如，对于不同场景的跟踪链(如不同格式的需求到不同语言的代码)，特征并不相同，从而导致机器学习只能在特定的场景下才能得到较好的效果，对于其它场景效果反而会下降. 尤其在数据集缺乏的情况下，ML 技术的效果将大打折扣.

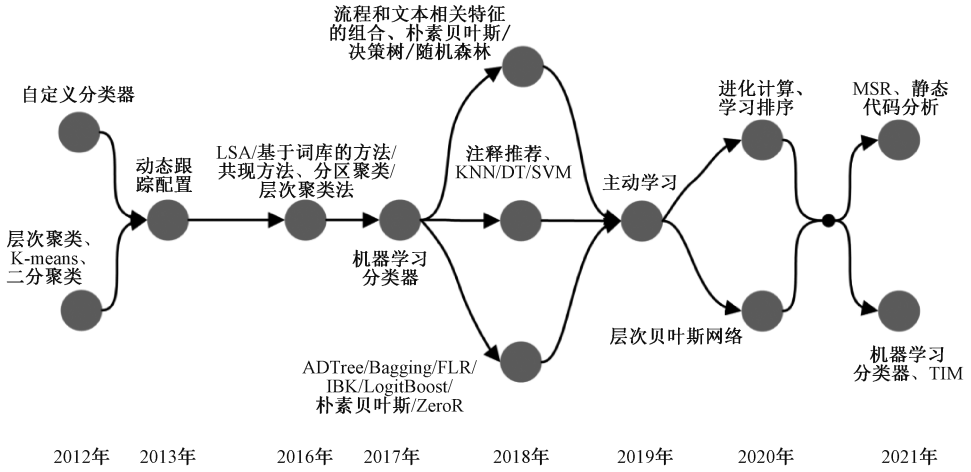

图 9 基于 ML 的跟踪链创建技术演变过程

3.1.4 基于 DL 的跟踪链创建

近年来，深度学习(DL)技术被越来越广泛地应用在跟踪链的创建中，它能够提高软件各制品之间跟踪链的准确性，并且解决了知识差距问题. 其基本原理是通过判断软件制品之间是否具有高度的文本相似性，从而进行跟踪链的识别和生成. 所采用的 DL 技术包括循环神经网络(Recurrent Neural Network，RNN)、前馈神经网络(Feedforward Neural Network，FNN)和词嵌入等. 基于 DL 的跟踪链创建包含四步：文本预处理、文本特征提取、神经网络模型构建与训练以及跟踪链生成. 表 5 统计了基于 DL 的跟踪链创建研究，并在图 10 中给出了这些新技术的演变过程.

为了解决需求跟踪链的多义问题，Wang 等人[19]提出了一种基于人工神经网络(Artificial Neural Networks，ANN)的方法，利用词对排序模型和簇排序模型，扩展了基于 IR 的方法. 词对排序模型利用词嵌入和 FNN，识别需求集合中的一词多义列表，然后由簇排序模型使用这些多义术语，更新术语到需求的链接. 词嵌入技术还被用于构建需求和代码之间的跟踪链，Hey 等[80]采用词嵌入和词移距离缩短了细粒度的需求与代码之间的语义差距. 陈磊等人[81]提出一种基于图挖掘拓展学习的无监督需求跟踪方法 GeT2Trace，通过挖掘软件制品文本网络中的语义特征来增强制品文本的语义表示，然后利用 Doc2Vec 和相似度模型来学习和比较制品的语义相似性.

针对特定领域的跟踪链，Guo 等人[82]开发了一种利用 RNN 模型生成制品之间的跟踪链网络，它包括两个阶段：(1)使用在大量领域文档上训练的无监督学习方法学习领域知识(如规则、需求、代码等)，并生成高维词向量，捕获每个词的语义分布和共现统计；(2)使用该领域已验证跟踪链的现有训练集来训练网络，比较两个制品的语义向量并输出它们之间链接的概率，以预测两个软件制品之间存在链接的可能性. 但该方法需要通过在训练集中识别和包含更具代表性的负样本来提高跟踪网络的精度. 鉴于软件项目往往包括技术术语以及特定领域

的专业术语,Liu 等人[83]提出了一种自动生成特定领域概念模型的方法.该方法首先创建与项目相关的查询来挖掘领域知识库,然后通过 AutoPhrase 工具自动提取概念,再采用 HiGrowth 框架识别同义关系、缩写关系、兄弟关系和祖先关系来创建概念模型,最后由此建立代码到设计之间的跟踪链.Lin 等人[84]提出了一个基于 Bert 的 Trace BERT(T-BERT)框架,用来跟踪代码到需求的链接,同时采用一个三步训练策略解决了训练数据集的问题.

Lam 等人[85]提出了一个名为 HyLoc 的模型,将深度神经网络(Deep Neural Network,DNN)与修正向量空间模型(revised Vector Space Model,rVSM)相结合,通过 rVSM 收集缺陷报告和源文件之间的文本相似性,再使用 DNN 将缺陷报告中的术语与潜在的不同代码标记以及源文件中的术语链接起来,以解决缺陷定位的词汇不匹配问题.DNN 和 rVSM 相互补充,能实现比单个模型更高的精度,精确率高于基线方法约 10%.

表 5　基于 DL 的跟踪链创建研究

方法/技术	跟踪链	文献
ANN	R-R、R-T	[19]
词嵌入、词移距离	R-C	[80]
GeT2Trace(Doc2Vec)	R-R、R-C、R-T、D-T、R-D、D-C	[81]
RNN	R-D	[82]
特定领域的概念模型、神经网络	C-Doc	[83]
Bert	R-C	[84]
DNN、rVSM	C-B	[85]

基于 DL 的跟踪链创建研究采用的新技术包括 Bert、Doc2Vec 和词嵌入技术等.与 ML 技术相比,DL 技术存在无需人工提取文本特征的优势.基于 DL 的跟踪链创建研究目前尚处于初步阶段,DL 技术还存在可解释性的问题,其所需数据集规模也更加庞大.

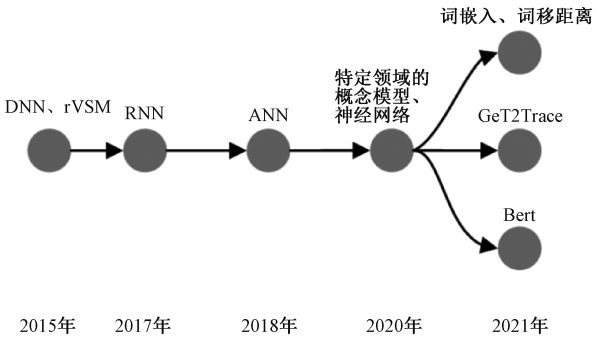

图 10　基于 DL 的跟踪链创建技术演变过程

3.1.5　启发法、模型法和其他

表 6 统计了一些基于启发法、模型法和其他技术的跟踪链创建研究.

(1) 启发法

启发法是针对所要跟踪的软件制品特点设计一系列启发式规则进行制品之间的匹配和映射.Ziftci 等人[86]提出了面向特征的需求跟踪分析(Feature Oriented Requirements Traceability Analysis,FORTA)方法,通过为每个软件特征创建场景,采用 AspectJ① 执行场景来收集执行轨迹单元并进行特征标记,最后创建需求到测试的跟踪矩阵.为了创建高可信系统的跟踪链,Mirakhorli[87-88]提出了一个以决策为中心的可跟踪框架(Decision-Centric Traceability,DCT),利用架构决策将架构紧密相关的需求链接到其具体实现片段(如组件、层、方法、变量、类).Gervasi 等人[89]提出一种基于亲和力的半自动需求跟踪方法,通过测量一对术语之间的亲和力,从而计算这些术语在相互关联需求中出现的可能性.

启发法适用于一些特殊类型的软件跟踪链.例如 Dagenais 等人[90]提出了一种启发式方法可以自动分析开源项目的文档,并且将相似代码术语(如 year())精确地链接到 API 中的特定代码元素(如 DateTime.year()).他们考虑了术语的上下文,并应用了一组过滤启发式方法,以确保引用外部代码元素的术语没有假链接.

采用启发法所构建的跟踪链精度较高,但需要根据不同制品特点设计跟踪规则,所以构建的多为特定种类的跟踪链,不适用于多种跟踪链的创建.

(2) 模型法

模型法通过构建跟踪元模型或对制品进行建模,为后续跟踪链的生成提供依据.

Ohashi 等[91]通过构建 UML 模型,定义制品的适当粒度,来解决需求到设计间的粒度差异问题.除 UML 外,形式化语言 LTL(Linear-Time Logic)和 AADL(Architecture Analysis & Design Language)以及 SysML(System Modeling Language)也可用来生成需求与设计之间的跟踪链[92-94].

Delater 等人[95]提出了一种动态捕获需求和代码之间的跟踪链方法.该方法的基础是跟踪信息模型 TIM,它以高精度和高召回率在制品之间创建了跟踪链,且易用性得到了提高.为了应对需求变更带来的影响,Duarte 等人[96]提出了关于需求跟踪的 BoK (Body of Knowledge)知识模型 TraceBoK,收

① "AspectJ," http://www.eclipse.org/aspectj/

表 6 启发法、模型法和其他方法

分类	方法/技术	跟踪链	文献
启发法	面向特征的需求跟踪分析 FORTA	R-T	[86]
	决策为中心的可跟踪框架 DCT	R-D	[87-88]
	基于亲和力的需求跟踪	R-D	[89]
	术语识别	C-Doc	[90]
模型法	UML 模型	R-D	[91]
	TIM、模型切片	R-D	[92-93]
	元模型	R-D	[94]
	TIM	R-C	[95]
	TraceBoK	R-C	[96]
	模型转换	R-C	[97]
	状态模型驱动	R-M	[98]
	语义建模	SS-B	[99]
其他	配置感知的程序分析方法 CAPA	R-C	[100]
	眼睛注视	R-C	[101-103]
	眼球跟踪、交互数据	C-Doc、R-C、R-D、R-T	[104]
	机器学习、Web 挖掘、查询增强	R-C	[4]

集软件跟踪方面的工程实践，帮助开发人员在软件产品的开发和维护中定义可跟踪性. 通过 TraceBoK，可以构建完整的需求跟踪文档，并实现知识改进. Vara 等[97]设计出一种名为 MeTAGeM-Trace 的框架，通过模型转换实现需求到代码的跟踪链生成. 此外，还有研究利用状态模型的逻辑检验[98]和语义网络模型的推理服务[99]，以此分析不同制品之间的跟踪链.

模型法需要在跟踪链创建前就将跟踪元素定义清楚，因此适用于定义明确、格式规范的软件制品，如需求、代码和架构. 对于诸如设计策略、测试文档此类难以在项目早期定义明确的软件制品，则很难实施.

(3) 其他方法

程序分析方法是有关代码与其他制品之间跟踪链创建的一项关键技术. 为了解决现有方法在分析不同软件变异性方面的局限性，Linsbauer 等人[100]使用配置感知的程序分析方法(Configuration-Aware Program Analysis，CAPA)来静态分析产品变体，创建系统依赖图，通过 ECCO (Extraction and Composition for Clone-and-Own)工具将软件特征映射到代码库.

Ali 等人[101-102]通过实证研究验证了人机交互方法的有效性. 他们首先利用眼球跟踪获得开发人员最关心的与需求相关的代码实体，再利用这些实体来创建需求与代码之间的跟踪链，能够提高需求到代码跟踪链的准确性. Walters 等[103]则使用眼球跟踪器来捕捉开发者在 Eclipse 集成开发环境中执行软件维护任务时的制品跟踪轨迹. 之后 Ahrens 等[104]提出了将眼球跟踪和交互数据结合的方法，通过跟踪开发人员与文档之间的交互方式来记录这些文档之间的凝视链接，并开发了一个眼球跟踪框架，使其能在软件开发过程中自动捕获凝视链接和凝视持续时间，为动态文档环境下眼球跟踪数据的记录提供软件支持. 上述人机交互方法可用于在项目早期实时构建跟踪链.

为解决跟踪链中的术语不匹配问题，Guo 等人[4]提出了融合三种技术(机器学习、Web 挖掘和查询增强)的混合式方法：①利用训练分类器从一组规则到需求的跟踪链中学习术语，以替换原始查询；②利用 IR 技术和 Web 挖掘学习术语，以替换原始查询；③使用领域本体增强查询. 但这些技术都有一定的使用限制，如训练集要求过大，术语定义是否明确等.

3.1.6 新技术/新方法的效果改进

本文选取一些在共同数据集上进行实验的跟踪链创建技术，讨论其相对基线方法的精度提升率. 如表 7 所示，P 表示精确率提升率，R 表示召回率提升率. 选取的技术包括 IR、ML 和 DL，数据集包括 EasyClinic、eTour、SMOS、CM1、iTrust 和 Gantt 项目. 这些方法的基线方法包括 VSM、LSI 和 JS.

IR 技术和平滑过滤器的结合[45-46]在 EasyClinic 数据集上较 VSM、LSI 和 JS，精确率和召回率均提升很多，达到 40% 左右. GeT2Trace[81]在 eTour 数据集上较 VSM 和 LSI 召回率提升幅度最高，分别达到 107% 和 90%，精确率提升约 25%；在 CM1 和 iTrust 数据集上较 LSI 精确率提高了 100% 以上. 该方法在 Gantt 数据集上的召回率也达到了

表 7　新技术/新方法的效果改进

新技术/新方法	基线方法	EasyClinic	eTour	SMOS	CM1	iTrust	Gantt
VSM/JS、关系主题建模[54]	VSM	—	+24%P+18%R	—	—	—	—
	JS	—	+19%P+14%R	—	—	—	—
VSM/LSI/JS、平滑过滤器[45-46]	VSM	+35%P+40%R	—	—	—	—	—
	LSI	+37%P+42%R	—	—	—	—	—
	JS	+43%P+40%R	—	—	—	—	—
显式语义分析[56]	VSM	—	−40%P +9%R	—	−20%P+24%R	−9%P +38%R	—
	LSI	—	+25%P+12%R	—	+98%P+64%R	+33%P+10%R	—
VSM、代码所有权[44]	VSM	—	+19%P	+33%P	—	—	—
VSM/JS、扩展语料库[49]	VSM	—	+20%P	+35%P	—	—	—
	JS	—	+28%P	+19%P	—	—	—
VSM、紧密度分析[63-64]	VSM	—	—	—	—	+6%P +4%R	+9%P +5%R
	LSI	—	—	—	—	+2%P +2%R	+5%P +2%R
	JS	—	—	—	—	+15%P+15%R	+5%P +2%R
VSM/JS、ConPOS[48]	VSM	—	—	—	—	+38%P+19%R	—
	JS	—	—	—	—	+44%P+14%R	—
VSM、紧密度分析、用户反馈[65]	VSM	—	—	—	—	+30%P+31%R	+17%P+19%R
	LSI	—	—	—	—	+21%P+24%R	+20%P+22%R
	JS	—	—	—	—	+27%P+30R	+42%P+38R
层次聚类、K-means、二分聚类[75]	VSM	—	+9%P+50%R		+3%P+98%R	+8%P+71%R	—
ANN[19]	VSM	—	—	—	+18%P+0%R	—	—
	LSI	—	—	—	+26%P+3%R	—	—
GeT2Trace(Doc2Vec)[81]	VSM	+22%P+12%R	+24%P+107%R	—	+19%P+22%R	+55%P−20%R	+19%P+47%R
	LSI	+8%P+10%R	+26%P+90%R	—	+151%P +22%R	+2100%P −5%R	+10%P+69%R

47%的提升. VSM、代码所有权[44]和 VSM/JS、扩展语料库[49]在 SMOS 数据集上的精确率提升幅度比 eTour 数据集大,在 SMOS 数据集较 VSM 提升了约 35%. 显式语义分析[56]在 CM1 数据集上较 LSI 有很大幅度的精确率和召回率提高,但相比 VSM 在精确率方面反而有下降. VSM、紧密度分析、用户反馈[65]在 iTrust 数据集上综合效果最好,精确率和召回率均有 20%～30%的提升. 如表 7 所示,不同新技术/新方法在多个数据集上的提升效果存在很大区别,部分技术在某些数据集上较基线方法提升较多,但在其他数据集上较特定基线方法也会有所降低,如 Get2Trace 在 iTrust 数据集上虽然精确率得到了很大幅度的提高,但召回率较 VSM 和 LSI 均有所下降.

3.2　跟踪链的维护

随着项目的维护和发展,添加新制品、删除原有制品以及更改现有制品,都可能导致跟踪信息过期,这时需要及时根据制品的变化更新去维护跟踪链. 针对这一问题,研究者们提出了跟踪链维护的解决方法,包括基于规则的维护、基于重构的维护、基于交互日志的维护. 通用的跟踪链维护步骤包括四步:变更检测、受影响跟踪链检测、跟踪链维护决策和维护方案实施.

在基于规则的维护工作中,Cleland-Huang 和 Mäder 等人[105]通过监控制品变更事件,再利用关联规则识别出与变更事件相关的跟踪链. Mäder 等[106]对于关联规则做了进一步的拓展. 该方法能够自动识别对现有跟踪链有影响的开发活动,然后使用规则来完成开发活动的必要更新. 该方法在跟踪关系的选择以及规则的扩展和定制上还需人工操作.

基于重构的维护由 Rahimi 等人提出[3,107],是对基于规则方法的改进. 他们首先总结了软件重构过程中可能发生的变更模式和变更场景,例如功能的增加、删除和修改、代码修改以及需求修改,针对每种场景设计不同的跟踪链维护规则. 通过重构检测工具和 VSM 技术检测变更场景,再根据维护规则自动更新跟踪链.

Hübner 等人提出了基于交互日志的维护方法——IL(Interaction Log)和 ILcom[67]. 相比已有的维护方法,其好处在于能够实时捕捉代码的变更. IL 方法首先在 IntelliJ 集成开发环境中捕捉交互事件并关联到需求,然后通过交互事件识别受此交互影响的代码并创建需求到代码的跟踪链,记录交互事件的元数据如交互频率、持续事件等,再通过元数据和源码结构删减一些错误的跟踪链来提升精确率

和召回率. ILcom 与 IL 方法类似,但对交互事件的捕捉做了改进,在 Eclipse 里通过提交消息和问题 ID 来实时地捕捉交互事件.

基于重构的维护对软件重构模式做了更深入的分析,涉及的变更事件更广,但目前只能处理离线代码. 基于交互日志的维护可以实时捕捉代码变更,但变更事件范围较少. 基于重构和交互日志仅支持需求到代码的跟踪链维护,基于规则的维护方法可支持更多类型的跟踪链,但需解决不同场景下的规则定制化和完整性问题.

跟踪链的维护至关重要,直接决定了软件项目后期阶段的制品跟踪是否有效. 然而,与跟踪链创建相比,跟踪链维护相关研究较少. 当前跟踪链维护研究的主要难点在于数据集缺乏. 由于跟踪链的维护需要初始跟踪链和制品数据以及变更后的跟踪链和制品数据,很多时候依赖的数据在项目中并不存在. 如何挖掘此类公开的数据集是实现和验证跟踪链维护的前提. 虽然已有文献通过代码提交消息[67]和版本变更[107]来识别代码变更,但也仅适用于与代码相关的跟踪链维护,其他类型的跟踪链维护研究则很难开展.

3.3 跟踪链的验证

由于创建或维护的跟踪链存在不正确性和不一致性,这就需要在创建和变更跟踪链后对其进行验证,包括基于源码结构的验证[108-109]、基于形式化方法的验证[94]、基于 ML 的验证[110].

Ghabi[108-109]等通过探索跟踪模式和代码中调用关系(如方法或函数)来识别需求与代码之间错误的跟踪链. 该方法能够以 85%～95%的精确率和 82%～96%的召回率检测出无效的跟踪链. Kuang 等人[111]研究了源码结构中的数据依赖关系在跟踪链验证中的作用,通过 5 个数据集上的实验表明,数据依赖关系与调用关系在跟踪链验证时可形成优势互补.

基于形式化方法的验证采用形式化描述软件制品之间的链接,再采用模型检验技术验证跟踪链的正确性. 例如,Goknil 等[94]开发了需求到架构的跟踪链验证器,将其集成到工具中. 他们定义了需求到架构的两种链接:满足和分配,并采用 LTL 和 AADL 分别描述需求和架构,最后通过形式化模型检验实现跟踪链的验证.

为了解决跟踪链验证的效率问题,Mills 等人[110]提出了跟踪链分类器(TRAceability lInk cLassifier,TRAIL)框架,用于自动验证不同级别跟踪链的有效性. 他们使用三个特性来验证跟踪链:余弦相似度、查询质量指标和文档统计数据,采用六个 ML 分类器试验 TRAIL:随机森林、K 近邻、多分类逻辑回归、朴素贝叶斯、SVM 和 Voted,结果表明随机森林的表现优于其他五种方法.

基于形式化方法的验证采用形式化规约和模型检验导致其效率不高,且仅适用于能转化为形式化规约的软件项目和制品. 基于源码结构的验证方法仅适用于与代码相关的跟踪链. 基于 ML 的验证方法虽然在效率上有所提升,但在准确率上低于形式化方法和源码结构方法.

4 软件跟踪链的应用研究

如何高效地使用软件跟踪链也是跟踪链自动化研究中的一大挑战[8]. 跟踪链在软件开发的各个活动中均扮演重要角色,包括需求管理(需求变更、需求检索、需求重用)、程序理解、软件演化与维护、项目管理、数据集构建、可视化应用等,如图 11 所示. 通过总结跟踪链的应用研究,分析现有应用问题和改进方案,可以充分发挥跟踪链的作用,极大提高多种软件开发活动的效率.

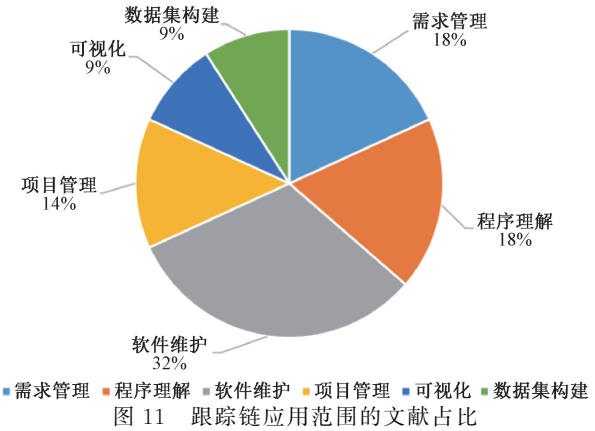

图 11 跟踪链应用范围的文献占比

(1)需求管理应用

Charrada 等[112]采用启发式规则构建需求到代码的跟踪链并将其用于过时需求的识别中. Saini 等[113]构建出的领域模型之间的跟踪链实现了需求模型的快速检索. Niu 等人[114]提出了跟踪链重构方法来管理实时需求. 为了更精确的定位需求变更后应该重构的代码,他们首先基于聚类的链接检索构建需求到代码的跟踪链,再根据跟踪链来定位需要重构的代码,最后利用特征化语义确定重构方案. 此外,Niu 等人[115]还提出一种灰链识别机制,并调研不同软件开发任务对同一需求跟踪的影响,通过实

验证明灰链能够很好地支持特定的需求变更和重用任务,未来还可将灰链用于跟踪链创建方法的验证与评估中.

(2)程序理解应用

Krüger 等人[116-117]探讨了跟踪链对程序理解的影响,并发现在代码中注释软件特征可以提高程序理解能力. Dit 等人[118]将跟踪链用于特征定位,提出了一种基于 TraceLab 的特征定位方法,解决了因数据集不同导致的方法不兼容、指标不统一等问题. Falessi 等人[119]提出了一个新的特征族,即需求到需求集(Requirements to Requirements Set, R2RS),旨在度量新需求和代码类关联的需求集合之间的语义相似性. 他们将 R2RS 度量标准与四种指标进行比较,包括变更的时间局限性、代码的直接相似性、复杂性度量、代码异味等度量标准,以展示和评估 R2RS 度量的有效性.

(3)软件维护应用

Mäder 等[36,120]探究了软件开发人员能否通过跟踪链从代码导航中获益,从而完成基本软件维护任务如更改软件特征、修复代码错误等. 他们在 Gantt 和 iTrust 两个软件项目上分别执行软件维护任务,通过"浏览无关代码"和"浏览相关代码"的百分比时间、三种导航类型的分布以及导航量占比来评估结果,最后证明了包含跟踪链的代码导航大大提高了变更处理的性能和质量. Mirakhorli 等[121]提出了一种基于 tTIM 的方法,该方法通过构建策略架构和代码的跟踪链,确保维护时的软件架构质量. Dagenais 等在前期代码与文档跟踪链创建工作[90]的基础上,采用模式匹配的方法 AdDoc 将跟踪链应用于用户文档等的变更[122]. 由于文档质量不同,AdDoc 在一些软件项目上的精确率和召回率差别很大.

此外,跟踪链还能被应用于安全攸关系统的维护,例如 Agrawal 等人[123]通过利用制品树中的跟踪链自动识别出在软件安全保障案例中的更新,从而确保此类系统不会因为代码或需求的变更出现安全问题. Mäder 等[124]通过实际案例评估了需求跟踪链对软件维护的影响,证明了需求跟踪链不仅能帮助开发人员提高软件维护的效率,还可以提高软件维护任务的质量,如减少代码质量退化等.

(4)项目管理应用

Cetin 等人[125]借助跟踪链图和图中心指标,识别出项目中有价值的开发人员,包括连接者和专家. 连接者为连接不同开发组的开发人员,而专家则为擅长开发某个特定模块的开发人员. 类似地,Sülün 等人[126-127]借助跟踪链图,相继开发出 RSTrace 和 RSTrace+工具,以识别出最适合审查代码的同行. 然而该方法还存在以下不足:①使用的评价指标并不适用所有审核场景,②不适用于在项目早期跟踪链图并不完善的情况.

(5)可视化应用

跟踪链的可视化通过交互式地探索多个制品之间的关系,帮助项目人员理解系统的整体结构,成为跟踪链的重要应用之一. 传统跟踪链可视化方法(如需求跟踪矩阵[128]、表格网格格式[129-130]、层次叶节点[131]、树状图[132-133]、Sunburst 和 Netmap[134])展示形式较为单一,且存在一些弊端,如需求跟踪矩阵或表格网格格式仅适用于小数据集验证,很难探索软件制品之间的相互关系,而基于图形的方法一次只能呈现两个制品之间的粗粒度视图.

Aung 等[135]提出了一种独立的交互式跟踪链可视化方法,该方法能够自然而直观地探索各种制品之间的相互关系,也能够使用外部数据自动创建跟踪链,还可以通过提供可视化跟踪链的探索空间来帮助开发人员理解需求变更和分析变更影响. DCTracVis[136]提供了更先进的跟踪链可视化技术,结合树状图和分层树技术来减少庞大的跟踪链图所带来的视觉混乱,可对跟踪链的全局结构和每个跟踪链的详细概述进行可视化,具有高度的可扩展性和交互性.

除了上述应用外,跟踪链还可用于构建软件开发任务的相关数据集. 例如,由于在软件评估领域的数据集数量不足且质量较低,Lin 等[137]和 Just 等[138]利用缺陷报告和代码修改之间的跟踪关系分别构造了真实缺陷的数据集 JaConTeBe 和 Defect4J,利用这些数据集,开发人员能够更加真实、精确地评估缺陷检测方法在实践中的有效性.

综上所述,跟踪链的应用范围越来越广泛,但目前仍然存在两个问题:(1)应用的软件项目规模较小;(2)由于跟踪链的创建必须在相应制品生成后才能实施,因此很多应用研究仍然局限在项目后期,如需求变更、程序理解、软件维护.

5　跟踪链自动化技术的评估

本节将介绍已有跟踪链自动化技术的评估工作,以及与之相关的数据集和评估指标,分析数据集和指标的适用场景,以此帮助研究人员在未来的工作中选择合适的技术、数据集和指标.

5.1 相关工作

在基于 IR 的跟踪链创建技术评估中，Mahmoud 等[139]使用来自不同应用领域的三个数据集，对三类语义赋能的 IR 方法进行了实验分析，包括语义增强、潜在语义和语义相关方法，用于评估语义信息检索方法在捕捉需求跟踪链方面的性能. Vale 等[140]采用 Shapiro-Wilk 实验方法，通过精确率、召回率、F-measure 和执行时间四个指标，在两个数据集上对比了五种 IR 技术（经典向量模型、LSI、神经网络、扩展布尔模型和 BM25），结果发现五种方法的执行时间相差不多，但在精确率和 F-Measure 等其他指标上存在差异.

此外，Qusef 等人[141]在两个软件系统 ArgoUML 和 eXVantage 上进行了两组控制实验，通过精确率、召回率和 P-value 三个指标，评估了三种测试到代码的跟踪链创建技术.

5.2 数据集

数据集在近年来跟踪链的自动化技术中发挥着重要作用. 研究人员通过不同的方式使用数据集开发、验证、评估新方法和新技术. 例如，基于机器学习和深度学习的跟踪链创建方法，均需使用训练集来训练模型，使用验证集来调整模型参数，再使用测试集测试模型的性能. 数据集本身的多样性和质量对研究结果的准确性、通用性和再现性有着重大影响. 我们对 2011－2021 年发表的相关工作中数据集进行了统计（图 12）.

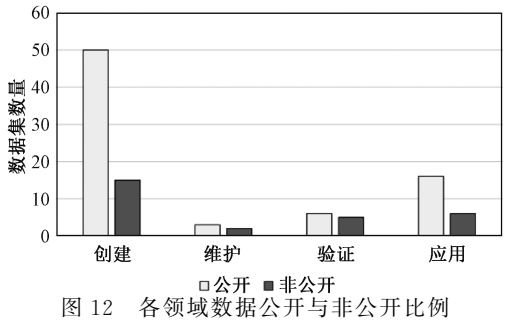

图 12 各领域数据公开与非公开比例

在跟踪链的创建中，我们共统计了 67 个数据集，其中 eTour、iTrust 和 EasyClinic 是使用最多的三个数据集. 如图 12 所示，在这些数据集中，包含 50 个公开数据集和 15 个非公开数据集，其中 80% 以上的数据集都与需求、代码制品相关. 跟踪链的维护包括 5 个数据集，跟踪链的验证则共有 11 个数据集. 在跟踪链的应用中，我们共统计了 22 个数据集，包含 73% 公开的数据集和 27% 的非公开数据集，iTrust 和 QT 3D Studio 是使用最多的数据集.

从图 12 中可以看到，当前的数据集大多集中在跟踪链的创建中，关于跟踪链维护、验证和应用的数据集则占比较少，为此类研究的推广和客观评估带来障碍. 此外，大多数据集仅涉及需求-代码跟踪链，其它类型的关键跟踪链（如代码-缺陷）的数据集则占比很少.

5.3 评估指标

为了量化评估不同跟踪链相关技术，研究人员提出了不同的评估指标（如表 8 所示）. 我们将详细阐述该领域研究中常用的评估指标，包括精确率（*Precision*）、召回率（*Recall*）、平均精确率（*Average Precision*，*AP*）和平均 AP 值（*Mean Average Precision*，*MAP*）等. 下面将对这些指标的定义及其计算方法逐一解释.

（1）精确率，是指被验证有效的链接与所有被算法识别出的链接之比. 具体计算公式如下：

$$Precision=\frac{TP}{TP+FP} \tag{1}$$

（2）召回率，是指被验证有效的链接与所有有效链接之比：

$$Recall=\frac{TP}{TP+FN} \tag{2}$$

其中 TP 表示被验证有效的链接集合，FP 表示被验证错误的链接集合，FN 表示有效却没有被算法识别的链接集合.

（3）*F-measure* 是结合了精确率和召回率的综合性能指标，是精确率和召回率的平均调和，并可用于比较不同实验的结果，$\beta(>0)$是参数，其中 F_1－*measure* 是最常用的形式（$\beta=1$）. *F-measure* 的计算公式如下：

$$F_\beta=\frac{(\beta^2+1)\times Precision}{\beta^2\cdot Precision+Recall} \tag{3}$$

（4）特异度（*Specificity*），用于计算实际无效链接和总体无效链接的比例. 其计算公式如下：

$$Specificity=\frac{TN}{TN+FP} \tag{4}$$

TN 表示算法识别出的实际无效链接集合.

（5）*AP* 用于表示某个类别中（如某个查询、需求等）被创建链接列表的排名准确程度，计算公式如下：

$$AP=\frac{\sum_{r=1}^{N}(Precision(r)\times isRelevant(r))}{|Relevantartifacts|} \tag{5}$$

其中，r 是链接列表当中目标制品的具体排名，*Precision*（r）表示其精确率，*isRelevant*（r）用于判断该链接是否相关，相关为 1，无关为 0，N 是方法所得

到的列表中链接总数.

(6) MAP 是所有类别的 AP 平均值,计算如下:

$$MAP=\frac{\sum_{q=1}^{Q}AP(q)}{q} \tag{6}$$

其中 q 是单个查询,Q 是查询总数.

(7) $DiffAR$[56],链接列表中 TP 和 FP 的平均相似度之间的差值. $DiffAR$ 值越高,表示该方法在区分有效链接和无效链接时划分的更清楚,即跟踪链推荐列表质量越高. 下面是 $DiffAR$ 的计算公式:

$$DiffAR=\frac{\sum_{(d,h)}sim(d,h)}{|L_T|}-\frac{\sum_{(d',h')}sim(d',h')}{|L_T|} \tag{7}$$

其中,$L=\{(d,h)\mid sim(d,h)\}$是该方法生成的一组跟踪链,L_T 是 L 中 TP 的子集,该子集中的一个链接被描述为(d, h),L_F 是 L 中 FP 的子集,该子集中的一个链接被描述为(d', h'),h 和 d 表示不同制品集 $H=\{h_1,\cdots, h_n\}$和 $D=\{d_1,\cdots, d_m\}$的制品,$sim(d,h)$是通过该方法得到的 d 和 h 之间的相似度,$sim(d',h')$是通过该方法得到的 d'和 h'之间的相似度.

(8)滞后系数(Lag)[56],用于表示在算法所得到的链接列表中相似度超过 TP 相似度的平均 FP 数,它表示在列表中每个正确链接之前出现的不正确链接的平均数,Lag 值越低,该跟踪链推荐列表的质量越高.

表 8　评估指标

指标	跟踪链创建	跟踪链维护	跟踪链验证	跟踪链应用	评估研究	文献
精确率	●	●	●	●	●	[1,3,4,6,9,18,19,40,43-48,50-52,54-57,60,61,64-69,73,76,81,82,86,90,95,98,99,106,107,109-112,114,115,119,122,124,129,133,136,141-144]
召回率	●	●	●	●	●	[1,3,4,6,9,18,19,40,43-48,50-52,54-57,60,61,64-69,73,81,82,86,90,95,97-99,106,107,109-112,114,115,119,122,129,133,136,141-144]
F-measure	●	●	●	●	●	[1,3,4,9,18,19,56,57,60,64-69,81,83,95,107,110,115,119,123,141,142,144]
执行时间	●	●		●	●	[1,94,116,120,124]
MAP	●				●	[4,6,48,51,55,56,58,60,64,65,72,82,83,139,142]
DiffAR	●				●	[51,56,139]
Lag	●				●	[56,58,139]
平均精确率	●					[6,48,55,64,65,72,82,133]
特异度	●					[18,144]
AUC	●					[58,142]
MCC	●					[43,73]

$$Lag=\frac{\sum_{(d,h)\in L}Lag(d,h)}{|L|} \tag{8}$$

(9)曲线下面积 AUC(Area Under the Curve)[142],该曲线一般为 ROC 曲线或 PR 曲线. ROC 曲线的横坐标为 FPR(伪阳性率),纵坐标为 TPR(真阳性率),PR 曲线的横坐标为召回率,纵坐标为精确率. 计算公式如下所示:

$$TPR=\frac{TP}{TP+FN} \tag{9}$$

$$FPR=\frac{FP}{FP+FN} \tag{10}$$

当 TPR 越大、FPR 越小时,方法效果越好. AUC 考虑了阈值的变动以及方法对于 TP 和 FP 的分类能力,在样本不平衡的情况下,依然能够对各个方法作出合理的评价. 当 FP 数量超过 TP 时,采用 PR 曲线更为合适. AUC 值越大,表示该方法性能越好.

(10)马修斯相关系数(Matthews Correlation Coefficient, MCC),是用于测量跟踪链分类的性能指标.

$$MCC=\frac{TP\times TN-FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{11}$$

其值在−1～1 之间,是描述实际跟踪链分类与预测分类之间的相关系数,越接近 1 越好.

(11)执行时间(Execution time)[140],表示生成链接的执行时间,用于评估各个方法的效率.

除上述指标外,Unterkalmsteiner 等人[145]提出了一些依赖专家主观打分的指标,如效率、准确性、一致性和自信度,以及一些在需求跟踪中用于评估需求复杂度的指标,如相关需求数、平均相关需求数和需求信息流,由于篇幅限制,在此不做赘述.

使用何种评价指标,取决于应用场景及研究人

员的关注点.在跟踪链创建的场景下,召回率适合用来评估获得的跟踪链是否全面,而精确率则适合评估获得的跟踪链是否准确.但精确率受实验样本比例分布影响,当不正确的链接数量过多,被预测为正确链接的数量也会越多,此时精确率就会下降,因此当实验样本分布不平衡时,使用精确率作为评价指标则不太合适,而应考虑叠加使用 *MCC*(常用来解决不平衡数据集的分类评价问题).如果要综合考虑召回率和精确率,则可使用 *F-Measure*.当 *F-Measure* 相差不大时,可考虑采用特异度帮助决策.*MAP*、*DiffAR* 和 *Lag* 常用于衡量一个跟踪链推荐列表的质量,它们能够反映链接推荐顺序所带来的差异,并在确认候选跟踪链时减轻人工筛选的工作量[31].在样本数据量比较大时,ROC 曲线会比较稳定,所以一般选择 ROC-AUC 来评价较为合适.

6 软件跟踪链相关工具

在软件跟踪过程中,研究人员通常会借助一些自动化或半自动化的工具来提高跟踪链创建、维护与验证效率,节约成本.然而,已有综述对此类工具集调研较少,本节通过分析软件跟踪链相关工具现状,为软件开发人员总结现有工具的目的、优势与缺点,便于其选择.根据工具的使用目的,我们将其分为面向软件跟踪链的工具(表 9)以及辅助性工具(表 10),辅助性工具并不仅限于软件跟踪链的创建、维护与验证过程,也可用作其它用途.

表 9 面向软件跟踪链的工具

工具名	目的	时间	开源	文献
FLAT	特征定位、代码识别、特征注释、跟踪可视化	2011 年	是	[149]
SCOTCH	创建测试和代码之间的跟踪链	2011 年	否	[150]
TraceLab	跟踪链创建、可跟踪性解决方案合成和评估	2012 年	是	[143]
过时需求识别原型工具	基于跟踪链识别过时需求	2012 年	否	[112]
TiQi	跟踪链查询	2012 年	否	[151]
Capra	软件跟踪链创建和管理	2016 年	是	[146]
MaTraCa	水平跟踪链创建	2016 年	是	[147]
FINALIsT	用于特征识别、特征定位、特征跟踪	2018 年	否	[152]
SAFA 工具集	可视化检索制品和链接的整个过程	2019 年	否	[123]
TCtracer	测试到代码的跟踪链创建	2020 年	否	[142]
RETRO	独立的生成需求跟踪矩阵	2020 年	是	[148]
RSTrace/ RSTrace+	基于跟踪链的代码评审推荐	2021 年	是	[127]

表 10 辅助性工具

工具名	目的	时间	是否开源	文献
TraCter	候选跟踪链聚类	2011 年	否	[154]
动态代码分析工具	代码依赖关系捕获和数据依赖关系	2015 年	是	[111]
Clafer	特征注释	2016 年	是	[153]
BUDGET	数据采样、提供可跟踪性的数据集	2017 年	是	[144]

Capra[146]是一个开源的、可扩展的软件跟踪链创建和管理工具,能够解决任意制品之间的跟踪问题.MaTraCa[147]是一个 Eclipse 插件,用于维护复杂应用程序中相同抽象级别实体(即代码、配置文件、html 表单等)之间的跟踪链.MaTraCa 专注于水平跟踪链的创建,能够指导开发人员根据依赖项所有者更新链接,但该工具仍需通过更多案例评估其有效性.RETRO[148]是一个可以独立生成跟踪矩阵的工具.RETRO 的核心是一个 IR 工具箱.该工具箱中的方法从 GUI 块中进行访问,解析输入的需求文档并生成候选链接列表,将其展现给需求分析人员.FLAT[149]是一套自动特征定位和文本跟踪工具,它集成了动态特征定位技术、特征注释技术和可视化技术,允许开发人员快速、轻松地识别与特征相关的代码,并保存注释供将来使用,但其仅支持 Java 代码,且在跟踪大规模软件项目制品时的性能和鲁棒性问题需要解决.SCOTCH[150]是一种基于动态切片和概念耦合开发的工具,用于创建测试和代码之间的跟踪链.TraceLab[57,143]是由 CoEST 提供的一款跟踪链自动创建开源工具.通过 TraceLab,研究人员可以使用可重用组件库和用户定义的组件库在可视化建模环境中设计和执行实验.TiQi[151]是一个提供自然语言查询跟踪链的原型工具,采用 Java 语言开发,可与已有的跟踪链创建工具或方法集成.FINALIsT[152]是一个半自动迭代工具,结合了 IR 技术和程序分析技术,通过识别、定位和记录特征代码为开发者提供帮助和支持,其优势在于能够处理海量的代码库,便于新手开发人员迅速上手,其缺点在于该工具的用户体验和稳定性并未评估.

TCtracer[142]是一种自动建立测试到代码链接的工具,与其它工作不同,TCtracer 在方法级别和类级别都能执行,允许开发者在测试和方法之间以及测试类之间建立链接.

在跟踪链的应用方面,RSTrace 及其升级版 RSTrace+[127]是由比尔肯大学开发的一款根据制品之间的跟踪链,推荐代码评审员的工具.该工具还提供了对 Github 的集成支持.然而该工具在大规模工业数据集上的测试还未完成.

在辅助性工具方面,动态代码分析工具[111]是由 Kuang 开发的一款用于捕获方法调用和数据依赖的集成式工具,它依赖于 JVMTI(Java 虚拟机工具接口),通过运行测试用例来捕获代码依赖关系和数据依赖关系,以便后期评估需求和代码之间的可跟踪性.Clafer 工具[153]在整个开发过程中都对代码进行了特征注释,因此可以通过访问版本控制历史构建特征到代码的链接,但该插件目前仅支持到 Java 代码的跟踪,且注释功能还不完善.BUDGET[144]是由 Zogaan 等人开发的开源数据采样工具,使用大数据分析和网络挖掘方法,实现了多种数据采样技术,根据用户定义的术语生成数据集,使研究人员能够在海量源文件的软件仓库中开展软件跟踪链研究,但该工具在采样的粒度上还需进一步细化.TraCter[154]是一个候选跟踪链的聚类工具,帮助开发人员执行跟踪任务.TraCter 使用高效的分割聚类算法并自动标记聚类,用层次树来显示复合簇,从而更灵活的执行复合聚类.

其他还有些实验性工具未列入表中,如构建用例和代码跟踪链的简单工具[36,120],目前只支持 Java 和 JSP 语言.此外,Niu 等人开发了一款面向需求重用、需求变更的跟踪链应用原型工具[115],该工具可与 Eclipse 很好的集成,但目前只支持需求到 Java 代码的跟踪链使用.Mirakhorli 等人[119]开发了一款用于捕获变更事件以及相关信息的原型工具,可集成到集成到 Visio Studio 和 Enterprise Architect 中.上述工具仅能作为原型系统,功能并不完整,因此并未被广泛使用.

近十年已有很多软件跟踪链相关的集成式和独立工具开发出来,如 TraceLab、MaTraCa、TCtrace、RSTrace 等,这些工具大多数为开源工具,帮助开发人员提升了跟踪软件制品的效率.经统计,半数以上的工具仅支持一种编程语言,约 1/3 的工具支持两种语言,支持两种以上语言的工具仅占 1/6.由于上述工具只通过实验或少量案例进行评估,功能完整性、性能和易用性等非功能指标尚未得到有效验证,其产业化、市场化进程缓慢.

7　关键问题与解决思路

7.1　关键问题

7.1.1　跟踪软件的复杂性问题

软件复杂性问题在软件跟踪链研究中经常被忽视[147].软件复杂性体现在横向复杂性和纵向复杂性两个方面.横向复杂性表现为代码的异构性以及编程范式、开发环境(如 IDE)和部署环境的多样性,纵向复杂性则表现为软件制品的多层抽象性,如不同粒度的 UML 类模型在需求、分析和设计阶段均可使用,为垂直跟踪链的创建和维护增加困难.随着互联网的发展,越来越多的软件和应用在开发时采用多种编程语言和集成多种环境,并由背景不同的涉众撰写文档[59,66,86,103].然而当前有关跟踪链的研究大多建立在统一的开发环境、单一的代码类型前提下[44,48-49,56,65,74,84],对软件复杂性问题关注不够,例如图 5 所展示的当前研究所关注的跟踪链大多为垂直跟踪链,如需求-代码、需求-设计、需求-测试,水平跟踪链创建、维护与验证的文献占比并不多,而与开发环境、编程语言紧密相关的水平跟踪链研究则更加缺乏.

7.1.2　跟踪链的粒度问题

现有很多有关跟踪链研究在跟踪制品的粒度上并不明确.以与代码相关的跟踪链创建为例,有些是针对模块和类级别(粗粒度)的跟踪,有些是针对方法、属性级别(细粒度)的跟踪.目前与代码相关的跟踪链创建中,粗粒度跟踪链占很大比例.通常粗粒度制品的跟踪相比细粒度的跟踪,花费的成本会降低,但细粒度的跟踪链能够提高方法的准确性.因此,如何根据软件项目的特点选择合适的制品粒度,实现跟踪链的扩展性和配置性,是目前有关跟踪链的自动化技术中遇到的一大挑战[32].

7.1.3　跟踪链的精度问题

表 7 中基于 ML 的跟踪链创建方法如层次聚类、K-means、二分聚类[75]在召回率上的改进效果良好,但精确率上提升不足.部分 IR 和代码分析技术集成的方法(如 VSM、紧密度分析、用户反馈[65]等)在精确率和召回率上的改进效果较好,其中 VSM/LSI/JS 和平滑过滤器的集成技术[45-46]、VSM、紧密度分析和用户反馈的集成技术[65]相比基线方法在精确率和召回率提升较高,均超过

20%.基于 DL 的方法(如 GeT2Trace[81])在不同数据集上的性能波动较大,且在一些数据集上精确率和召回率改进幅度也差别很大.在 eTour 和 Gantt 数据集上,召回率提升度超过了精确率 2～6 倍;在 iTrust 数据集上,精确率的提高却伴随着召回率的下降.基于 DL 的方法受数据集类型和质量影响较大,在采用自然语言描述的软件制品上效果较好,而在与代码相关的跟踪链上效果不显著.在跟踪链创建过程中,如果仅仅依赖智能化技术,会导致一些与代码相关的软件制品的关键特征被忽略,如制品的所有权[44]、方法调用和数据依赖关系[64]等.

7.1.4 跟踪链的类型受限问题

由图 5 可知,目前所研究的被跟踪软件制品大多集中在需求[41]、代码[64]和测试[6],这导致所识别的跟踪链种类非常有限.虽然也有部分研究专注于通用类型软件制品之间的跟踪链,但当类型变得通用之后,其准确度也会下降[77,110].例如代码依赖性方法只能应用在与代码相关的软件制品中,对于其他类型的目标制品则无法应用或效果降低,因此如何在拓展跟踪链类型的同时,保证链接的精确率和召回率等指标,也是当前软件跟踪链需要解决的一大问题.

7.1.5 跟踪链的验证效率问题

当前关于软件跟踪链的研究大多还专注于跟踪链的创建,在跟踪链的自动化验证上关注较少.虽然已有工作尝试将机器学习、信息检索技术应用于跟踪链的验证上,但仍然没有实现全自动化[110].此类方法缺少标准化、量化的验证标准,以及自动化技术和工具[141],严重制约了跟踪链验证的效率.

7.1.6 跟踪链的应用规模和时间问题

很多跟踪链应用的研究工作所采用的项目规模比较小[114-115,119-120,125,137-138],而将基于新技术所创建的跟踪链应用到大规模企业级项目的研究更少,其实际应用效果需要进一步的验证.此外,目前大多数跟踪链的创建都是采用离线恢复的方式,即在需求、设计、代码、测试等都开发完毕后,再开始生成跟踪链[48,51,56,64-65,72,74,78,103],从而导致很多跟踪链只能局限于项目后期,影响了跟踪链的应用前景.

7.1.7 跟踪链工具评估不全面问题

软件跟踪链领域已开发出很多自动或半自动化的工具或工具集,然而对于这些工具集的评估当前仅停留在实验和少量案例验证阶段,很少从产品的角度去评估跟踪链工具的功能完整性、性能、可用性、可靠性、易用性、用户满意度等,导致缺乏在工业界被广泛使用的跟踪链工具.这些工具大多支持的代码语言单一,严重限制了这些工具的使用范围和客观评估.

7.2 解决思路

7.2.1 构建软件制品间的水平跟踪链

针对横向复杂性和纵向复杂性,目前已经设计出许多开发软件和插件,为解决上述两方面的软件复杂性问题提供了有效措施[147].现有许多工作都集中在解决纵向复杂性,解决方法较为完善,可在不同的分析设计阶段,撰写不同级别的文档,进行递推性的纵向跟踪链识别和创建.然而,横向复杂性的解决方法依然存在一些问题.对于横向复杂性,目前使用较多的方法是逻辑耦合[155].该方法能够挖掘出软件制品之间可能同时被修改的链接,并建立制品之间的水平跟踪链.然而,逻辑耦合在使用时有诸多约束条件,未来可考虑结合元模型的方法,通过构建一个元模型来链接不同软件制品如代码、UML 和单元测试,并在元模型中添加传播规则以识别传递性链接.

7.2.2 可扩展、可配置的跟踪链自动化技术

不同粒度制品之间的跟踪链粒度和定义不同,因此需要在设计新方法或新技术时考虑多种粒度的可能[18],对于不同粒度的跟踪链,通过构建跟踪信息模型进行准确定义.例如文献[107]在设计代码变更模式时就同时考虑了类级别和方法级别的代码变更,再如文献[94]定义了需求到架构的两种跟踪链和不同级别.对于基于机器学习或深度学习的方法,则需在数据集构建时获取尽量多的信息和标签,便于后期模型在不同粒度上的学习.在技术评估时,应将项目规模、代码特点、执行时间、精确率、召回率等多个指标纳入考虑范围,权衡成本和效益,探索合适粒度的跟踪链自动化配置.

7.2.3 传统方法与人工智能技术的融合

未来工作中可以考虑深度结合传统方法(如模型法、程序分析方法等)与一些人工智能跟踪技术,来提高跟踪链的准确性[4],例如将模型法与机器学习方法相结合[69].在构建概念模型时,首先收集特定领域文档的语料库,创建相关搜索查询,然后分析文本信息和代码类层次结构,再基于给定概念模型,利用机器学习或深度学习算法来评估其相关性,从而生成相应的跟踪链.再如,大量研究发现代码中的数据依赖关系有助于理解软件制品之间的可跟踪性[64],将现有学习模型与代码依赖性分析方法充分融合将成为代码跟踪的趋势.在未来,还可以尝试使

用不同类型的代码依赖组成的代码模式来提高软件跟踪链的精确率.

7.2.4　利用中介制品创建多类型的跟踪链

由于软件制品间的跟踪链是可传递的,目前已有方法利用多种软件制品之间的关联,通过增加中介制品来实现多种类型的制品间的链接识别[57,118].未来可以考虑将中介制品方法与机器学习以及深度学习模型紧密融合.首先利用预训练模型,计算源制品和中介制品之间的相似度,并选取相对于源制品排名靠前的中介制品,再分别计算这些中介制品和目标制品之间的相似度,然后通过综合计算,得出三种制品之间的关系,生成多条跟踪链,进而确定源制品与目标制品之间的链接.

7.2.5　跟踪链的交互式验证

对于跟踪链的验证,需制定量化的跟踪链验证和评估指标,以支持跨方法、跨项目和跨技术的准确比较.未来可以对代码注释程度、用例与测试用例间的关联程度等指标进行量化,同时引入动态反馈机制,使跟踪链不仅能记录软件制品的链接,还可以接收项目涉众的动态反馈(如制品的变更、跟踪链的有效性等信息),从而设计一种交互式的跟踪链验证和演化机制,以提高验证效率与生产力.

7.2.6　跟踪链的实时获取和相关代码开源

针对软件跟踪链仅能应用于项目中后期问题,有必要深入研究跟踪链的实时获取技术,以便在软件制品生成之初就能实时地构建制品之间的跟踪链.实时获取技术需要同时考虑对软件系统本身性能的影响,对制品数据按冷热启动分类.目前已有一些工作研究跟踪链的实时获取技术[103-104],但仍然存在由于人工操作不当而导致跟踪链质量降低的问题.针对跟踪链的应用规模较小问题,需要利用软件工程开源社区的力量,增加新技术和新方法的使用用户,大量及时的用户反馈才能促进跟踪链自动化技术的提升,从而扩大应用规模.此外,还需制定量化的跟踪链软件产品的功能性和非功能性指标,全面、客观地评价已有工具的功能完整性、性能、用户满意度等,为工具的使用提供指导.

8　总　结

本文介绍了当前软件跟踪链领域的一些自动化技术研究,使用了综述的方式,从跟踪链的创建、维护、验证、应用、评估研究、工具几个方面,梳理并总结了当前的研究进展,并讨论了近年来火热的人工智能技术的应用.同时,本文对当前跟踪链自动化研究中所存在的关键问题和技术难点进行了阐述,并且给出了未来展望.根据综述表明:(1)由于当前软件开发的总体形势趋于复杂,软件维护的成本也在增加,软件跟踪链的创建一直是学术界关注的重点,有关跟踪链的维护、验证和应用研究则需得到更多的重视;(2)需求到代码的跟踪链是研究人员最关心的类型,其次是需求到设计、设计到代码,而如代码/数据到模型、屏幕截图到缺陷等一些小众的跟踪链也开始进入研究人员的视野;(3)随着人工智能的不断发展,基于人工智能技术的跟踪链相关研究数量也在逐步增加,如朴素贝叶斯、SVM、Bert、Doc2Vec、RNN 等智能技术已被广泛应用于跟踪链的创建、维护与验证;(4)在跟踪链的创建中,仅依靠信息检索和人工智能技术,很难取得很好的效果,未来应将信息检索、机器学习或深度学习与传统启发法、模型法等进行结合,弥补人工智能技术和传统方法中所存在的不足,进一步提高跟踪链质量;(5)未来应将复杂环境下、跨平台、跨语言的跟踪链自动化技术研究提上日程.

参　考　文　献

[1] CoEST: Center of excellence for software traceability, http://www.CoEST.org 2021, 7, 24

[2] Mei Hong, Wang Qian-Xiang, Zhang Lu, et al. Software analysis:a road map. Chinese Journal of computers, 2009, 32(9): 1697-1710.(in Chinese)
(梅宏, 王千祥, 张路, 等. 软件分析技术进展. 计算机学报, 2009, 32(9): 1697-1710.)

[3] Rahimi M, Goss W, Cleland-Huang J. Evolving requirements-to-code trace links across versions of a software system//Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution. Raleigh, USA, 2016: 99-109

[4] Guo J, Gibiec M, Cleland-Huang J. Tackling the term-mismatch problem in automated trace retrieval. Empirical Softwasre Engineering, 2017, 22(3): 1103-1142

[5] Youm K C, Ahn J, Kim J, et al. Bug localization based on code change histories and bug reports//Proceedings of the 2015 Asia-Pacific Software Engineering Conference. New Delhi, India, 2015: 190-197

[6] Moran K, Palacio D N, Bernal-Cárdenas C, et al. Improving the effectiveness of traceability link recovery using hierarchical bayesian networks//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Seoul, South Korea, 2020: 873-885

[7] Li Yin, Li Juan, Li Ming-Shu. Research on dynamic require-

ment traceability method and traces precision. Journal of Software, 2009, 20(2): 177-192. (in Chinese)
(李引, 李娟, 李明树. 动态需求跟踪方法及跟踪精度问题研究. 软件学报, 2009, 20(2): 177-192.)

[8] Gotel O, Cleland-Huang J, Hayes J H, et al. The quest for ubiquity: a roadmap for software and systems traceability research//Proceedings of the 2012 20th IEEE International Requirements Engineering Conference. Chicago, USA, 2012:71-80

[9] Chen X, Hosking J, Grundy J. A combination approach for enhancing automated traceability//Proceedings of the 2011 33rd International Conference on Software Engineering. Honolulu, USA, 2011:912-915

[10] Winkler S, von Pilgrim J. A survey of traceability in requirements engineering and model-driven development. Software & Systems Modeling, 2010, 9(4): 529-565

[11] Murugesan A, Whalen M W, Ghassabani E, et al. Complete traceability for requirements in satisfaction arguments//Proceedings of the 2016 IEEE 24th International Requirements Engineering Conference (RE). Beijing, China, 2016: 359-364

[12] Park S, Kim H, Ko Y, et al. Implementation of an efficient requirements analysis supporting system using similarity measure techniques. Information and Software Technology, 2000, 42(6): 429-438

[13] Çetin H A, Tüzün E. Identifying key developers using artifact traceability graphs//Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering. New York, USA, 2020: 51-60

[14] Antoniol G, Canfora G, Casazza G, et al. Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, 2002, 28(10): 970-983

[15] Cleland-Huang J, Settimi R, Duan C, et al. Utilizing supporting evidence to improve dynamic requirements traceability//Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE). Paris, France, 2005: 135-144

[16] Gotel O C Z, Finkelstein C W. An analysis of the requirements traceability problem//Proceedings of the IEEE International Conference on Requirements Engineering. Colorado Springs, USA, 1994: 94-101

[17] Witten I H, Paynter G W, Frank E, et al. Kea: Practical automatic keyphrase extraction//Proceedings of the Fourth ACM Conference on Digital Libraries. Berkeley, USA, 1999: 254-255

[18] Mirakhorli M, Shin Y, Cleland-Huang J, et al. A tactic-centric approach for automating traceability of quality concerns. //Proceedings of the 2012 34th International Conference on Software Engineering Zurich, Switzerland, 2012: 639-649

[19] Wang W, Niu N, Liu H, et al. Enhancing automated requirements traceability by resolving polysemy//Proceedings of the 2018 IEEE 26th International Requirements Engineering Conference. Banff, Canada, 2018: 40-51

[20] Hey T. INDIRECT: intent-driven requirements-to-code traceability//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering. Montreal, Canada, 2019: 190-191

[21] De Lucia A, Fasano F, Oliveto R, et al. Enhancing an artefact management system with traceability recovery features //Proceedings of the 20th IEEE International Conference on Software Maintenance. Chicago, USA, 2004: 306-315

[22] Dekhtyar A, Hayes J H, Sundaram S, et al. Technique integration for requirements assessment//Proceedings of the 15th IEEE International Requirements Engineering Conference. Delhi, India, 2007: 141-150

[23] Mahmoud A. Toward an effective automated tracing process//Proceedings of the 2012 20th IEEE International Conference on Program Comprehension. Passau, Germany, 2012: 269-272

[24] Lucia D. Information retrieval models for recovering traceability links between code and documentation//Proceedings of the 2000 International Conference on Software Maintenance. San Jose, USA, 2000: 40-49

[25] Torkar R, Gorschek T, Feldt R, et al. Requirements traceability: a systematic review and industry case study. International Journal of Software Engineering and Knowledge Engineering, 2012, 22(03): 385-433

[26] Borg M, Runeson P. IR in software traceability: from a bird's eye view//Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Baltimore, USA, 2013: 243-246

[27] Borg M, Runeson P, Ardö A. Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering, 2014, 19(6): 1565-1616

[28] Javed M A, Zdun U. A systematic literature review of traceability approaches between software architecture and source code//Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. London, England United Kingdom, 2014: 1-10

[29] Parizi R M, Lee S P, Dabbagh M. Achievements and challenges in state-of-the-art software traceability between test and code artifacts. IEEE Transactions on Reliability, 2014, 63(4):913-926

[30] Cleland-Huang J, Gotel O C Z, Huffman Hayes J, et al. Software traceability: Trends and future directions//Proceedings of the Future of Software Engineering Proceedings. Hyderabad, India, 2014: 55-69

[31] Hu Cheng-Hai, Peng Rong, Wang Bang-Chao. A surveys of requirement tracking method based on information retrieval. Computer Applications and Software, 2017, 34(10): 20-28. (in Chinese)
(胡成海,彭蓉,王帮超.基于信息检索的需求跟踪方法综述.计算机应用与软件, 2017, 34(10): 20-28.)

[32] Wang B, Peng R, Li Y, et al. Requirements traceability technologies and technology transfer decision support: A systematic review. Journal of Systems and Software, 2018, 146: 59-79

[33] Aung T W W, Huo H, Sui Y. A literature review of automatic traceability links recovery for software change impact analysis//Proceedings of the 28th International Conference on Program Comprehension. Seoul, Korea, 2020: 14-24

[34] Tian F, Wang T, Liang P, et al. The impact of traceability on software maintenance and evolution: A mapping study. Journal of Software: Evolution and Process, 2021, 33(10): 1-31

[35] Rempel P, Mäder P. A quality model for the systematic assessment of requirements traceability//Proceedings of the 2015 IEEE 23rd International Requirements Engineering Conference. Ottawa, Canada, 2015: 176-185

[36] Mäder P, Egyed A. Do software engineers benefit from source code navigation with traceability? an experiment in software change management//Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. Lawrence, USA, 2011: 444-447

[37] Kitchenham B A, Dyba T, Jorgensen M. Evidence-based software engineering//Proceedings of the 26th International Conference on Software Engineering. Edinburgh, UK, 2004: 273-281

[38] Zhang H, Babar M A, Tell P. Identifying relevant studies in software engineering. Information and Software Technology, 2011, 53(6): 625-637

[39] Kitchenham B, Pretorius R, Budgen D, et al. Systematic literature reviews in software engineering-a tertiary study. Information and Software Technology, 2010. 52(8): 792-805

[40] Wang Jin-Shui, Weng Wei, Peng Xin. Recovering traceability links using syntactic analysis. Journal of Computer Research and Development, 2015, 52(3):729-737. (in Chinese)
(王金水, 翁伟, 彭鑫. 一种基于句法分析的跟踪关系恢复方法. 计算机研究与发展, 2015, 52(3):729-737.)

[41] Wang Fei, Huang Zhi-Qiu, Yang Zhi-Bin, Kang Shuang-Long, Shen Guo-Hua, Chen Guang-Ying. A requirements traceability approach for safety-critical embedded system. Chinese Journal Of Computers, 2018, 41(3): 652-669. (in Chinese)
(王飞,黄志球,杨志斌,阚双龙,沈国华,陈光颖. 一种安全攸关嵌入式系统需求追踪方法. 计算机学报, 2018, 41(3): 652-669.)

[42] Schlutter A, Vogelsang A. Trace link recovery using semantic relation graphs and spreading activation//Proceedings of 2020 IEEE 28th International Requirements Engineering Conference. Zurich, Switzerland, 2020: 20-31

[43] Lapeña R, Pérez F, Cetina C, et al. Improving traceability links recovery in process models through an ontological expansion of requirements//Proceedings of the International Conference on Advanced Information Systems Engineering. Cairo, Egypt, 2019: 261-275

[44] Diaz D, Bavota G, Marcus A, et al. Using code ownership to improve ir-based traceability link recovery//Proceedings of the 2013 21st International Conference on Program Comprehension. San Francisco, USA, 2013: 123-132

[45] De Lucia A, Di Penta M, Oliveto R, et al. Improving ir-based traceability recovery using smoothing filters//Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension. Kingston, Canada, 2011: 21-30

[46] De Lucia A, Di Penta M, Oliveto R, et al. Applying a smoothing filter to improve ir-based traceability recovery processes: An empirical investigation. Information and Software Technology, 2013, 55(4): 741-754

[47] Capobianco G, Lucia A D, Oliveto R, et al. Improving IR-based traceability recovery via noun-based indexing of software artifacts. Journal of Software: Evolution and Process, 2013, 25(7): 743-762

[48] Ali N, Cai H, Hamou-Lhadj A, et al. Exploiting Parts-of-Speech for effective automated requirements traceability. Information and Software Technology, 2019, 106: 126-141

[49] Dasgupta T, Grechanik M, Moritz E, et al. Enhancing software traceability by automatically expanding corpora with relevant documentation//Proceedings of the 2013 IEEE International Conference on Software Maintenance. Eindhoven, Netherlands, 2013: 320-329

[50] Wu R, Zhang H, Kim S, et al. Relink: recovering links between bugs and changes//Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. Szeged, Hungary, 2011: 15-25

[51] Mahmoud A, Niu N. Supporting requirements traceability through refactoring//Proceedings of the 2013 21st IEEE International Requirements Engineering Conference. Rio de Janeiro, Brazil, 2013: 32-41

[52] Qusef A, Bavota G, Oliveto R, et al. Recovering test-to-code traceability using slicing and textual analysis. Journal of Systems and Software, 2014, 88: 147-168

[53] Heck P, Zaidman A. Horizontal traceability for just-in-time requirements: the case for open source feature requests. Journal of Software: Evolution and Process, 2014, 26(12): 1280-1296

[54] Gethers M, Oliveto R, Poshyvanyk D, et al. On integrating orthogonal information retrieval methods to improve traceability recovery//Proceedings of the 2011 27th IEEE International Conference on Software Maintenance. Williamsburg, USA, 2011: 133-142

[55] Dietrich T, Cleland-Huang J, Shin Y. Learning effective query transformations for enhanced requirements trace retrieval//Proceedings of 2013 28th IEEE/ACM International Conference on Automated Software Engineering. Silicon Valley, USA, 2013: 586-591

[56] Mahmoud A, Niu N, Xu S. A semantic relatedness approach

for traceability link recovery//Proceedings of the 2012 20th IEEE International Conference on Program Comprehension. Passau, Germany, 2012: 183-192

[57] Nishikawa K, Washizaki H, Fukazawa Y, et al. Recovering transitive traceability links among software artifacts//Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution. Bremen, Germany, 2015: 576-580

[58] Rodriguez A D, Cleland-Huang J, Falessi D. Leveraging intermediate artifacts to improve automated trace link retrieval//Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution. Luxembourg, 2021: 81-92

[59] Zhang Y, Lo D, Xia X, et al. Inferring links between concerns and methods with multi-abstraction vector space model//Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution. Raleigh, USA, 2016: 110-121

[60] Ali N, Guéhéneuc Y G, Antoniol G. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. IEEE Transactions on Software Engineering, 2013,39(5):725-741

[61] Ali N. Trustrace: improving automated trace retrieval through resource trust analysis//Proceedings of the International Conference on Program Comprehension. Kingston, Canada, 2011:230-233

[62] Guo J, Monaikul N, Plepel C, et al. Towards an intelligent domain-specific traceability solution//Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. Vasteras, Sweden, 2014:755-766

[63] Kuang H, Mäder P, Hu H, et al. Do data dependencies in source code complement call dependencies for understanding requirements traceability? //Proceedings of the 2012 28th IEEE International Conference on Software Maintenance. Trento, Italy, 2012:181-190

[64] Kuang H, Nie J, Hu H, et al. Analyzing closeness of code dependencies for improving ir-based traceability recovery// Proceedings of the 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering. Klagenfurt, Austria, 2017:68-78

[65] Kuang H, Gao H, Hu H, et al. Using frugal user feedback with closeness analysis on code to improve ir-based traceability recovery//Proceedings of the 2019 IEEE/ACM 27th International Conference on Program Comprehension. Montreal, Canada, 2019:369-379

[66] Tsuchiya R, Washizaki H, Fukazawa Y, et al. Interactive recovery of requirements traceability links using user feedback and configuration management logs//Proceedings of the International Conference on Advanced Information Systems Engineering. Stockholm, Sweden, 2015:247-262

[67] Hübner P, Paech B. Interaction-based creation and maintenance of continuously usable trace links between requirements and source code. Empirical Software Engineering, 2020, 25(5): 4350-4377

[68] Abukwaik H, Burger A, Andam B K, et al. Semi-automated feature traceability with embedded annotations //Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution. Madrid, Spain, 2018:529-533

[69] Rath M, Rendall J, Guo J L C, et al. Traceability in the wild: automatically augmenting incomplete trace links//Proceedings of the IEEE/ACM 40th International Conference on Software Engineering. Gothenburg, Sweden, 2018:834-845

[70] Mills C, Escobar-Avila J, Bhattacharya A, et al. Tracing with less data: Active learning for classification-based traceability link recovery//Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution. Cleveland, USA, 2019:103-113

[71] Mahmoud A, Williams G. Detecting, classifying, tracing non-functional software requirements. Requirements Engineering, 2016, 21(3):357-381

[72] Lohar S, Amornborvornwong S, Zisman A, et al. Improving trace accuracy through data-driven configuration and composition of tracing features//Proceedings of the 2013 9th ACM Joint Meeting on European Software Engineering Conference and Symposium on Foundations of Software Engineering. Saint Petersburg, Russia, 2013:378-388

[73] Marcén A C, Lapeña R, Pastor O, et al. Traceability link recovery between requirements and models using an evolutionary algorithm guided by a learning to rank algorithm: Train control and management case. Journal of Systems and Software, 2020, 163:1-24

[74] Saini R, Mussbacher G, Guo J L C, et al. Automated traceability for domain modelling decisions empowered by artificial intelligence//Proceedings of the 2021 IEEE 29th International Requirements Engineering Conference. Notre Dame, USA, 2021:173-184

[75] Niu N, Mahmoud A. Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited//Proceedings of the 2012 20th IEEE International Requirements Engineering Conference. Chicago, USA, 2012:81-90

[76] Mezghani M, Kang J, Kang E B, et al. Clustering for traceability managing in system specifications//Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference. Jeju Island, South Korea South, 2019:257-264

[77] Falessi D, Di Penta M, Canfora G, et al. Estimating the number of remaining links in traceability recovery//Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering. Montpellier, France, 2018:953-953

[78] Mills C. Automating traceability link recovery through classification//Proceedings of the 2017 11th ACM Joint Meeting on European Software Engineering Conference and Symposium on Foundations of Software Engineering. Paderborn, Germany, 2017:1068-1070

[79] Njomou A T, Africa A J B, Adams B, et al. Msr4ml: Reconstructing artifact traceability in machine learning repositories//Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering. Honolulu, USA, 2021:536-540

[80] Hey T, Chen F, Weigelt S, et al. Improving traceability link recovery using fine-grained requirements-to-code relations//Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution. Luxembourg, 2021: 12-22

[81] Chen Lei, Wang Dan-Dan, Wang Qing, Shi Lin. Enhancing requirements traceability recovery viaa graphMining based expansion learning. Journal of Computer Research and Development, 2021, 58(4): 777-793. (in Chinese)
(陈磊, 王丹丹, 王青, 石琳. 基于图挖掘扩展学习的增强需求跟踪恢复方法. 计算机研究与发展. 2021, 58(4): 777-793.)

[82] Guo J, Cheng J, Cleland-Huang J. Semantically enhanced software traceability using deep learning techniques//Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering. Buenos Aires, Argentina, 2017: 3-14

[83] Liu Y, Lin J, Zeng Q, et al. Towards semantically guided traceability//Proceedings of the 2020 IEEE 28th International Requirements Engineering Conference. Zurich, Switzerland, 2020: 328-333

[84] Lin J, Liu Y, Zeng Q, et al. Traceability transformed: generating more accurate links with pre-trained bert models//Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering. Madrid, ES, 2021: 324-335

[85] Lam A N, Nguyen A T, Nguyen H A, et al. Combining deep learning with information retrieval to localize buggy files for bug reports//Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering. Lincoln, USA, 2015: 476-481

[86] Ziftci C, Krueger I. Tracing requirements to tests with high precision and recall//Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. Lawrence, USA, 2011: 472-475

[87] Mirakhorli M. Tracing architecturally significant requirements: a decision-centric approach//Proceedings of the 33rd International Conference on Software Engineering. Honolulu, USA, 2011: 1126-1127

[88] Mirakhorli M, Cleland-Huang J. Tracing architectural concerns in high assurance systems//Proceedings of the 33rd International Conference on Software Engineering. Honolulu, USA, 2011: 908-911

[89] Gervasi V, Zowghi D. Supporting traceability through affinity mining//Proceedings of the 2014 IEEE 22nd International Requirements Engineering Conference (RE). Karlskrona, Sweden, 2014: 143-152

[90] Dagenais B, Robillard M P. Recovering traceability links between an API and its learning resources//Proceedings of the 2012 34th International Conference on Software Engineering (ICSE). Zurich, Switzerland, 2012: 47-57

[91] Ohashi K, Kurihara H, Tananaka Y, et al. A means of establishing traceability based on a UML model in business application development//Proceedings of the2011 IEEE 19th International Requirements Engineering Conference. Trento, 2011: 279-284

[92] Nejati S, Sabetzadeh M, Falessi D, et al. A SysML-based approach to traceability management and design slicing in support of safety certification: framework, tool support, and case studies. Information and Software Technology, 2012, 54(6): 569-590

[93] Falessi D, Nejati S, Sabetzadeh M, et al. SafeSlice: A model slicing and design safety inspection tool for SysML//Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. Szeged, Hungary, 2011: 460-463

[94] Goknil A, Kurtev I, Van Den Berg K. Generation and validation of traces between requirements and architecture based on formal trace semantics. Journal of Systems and Software, 2014, 88: 112-137

[95] Delater A, Paech B. Tracing requirements and source code during software development: An empirical study//Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Baltimore, USA, 2013: 25-34

[96] Duarte A M D, Duarte D, Thiry M. TraceBoK: Toward a software requirements traceability body of knowledge//Proceedings of the 2016 IEEE 24th International Requirements Engineering Conference (RE). Beijing, China, 2016: 236-245

[97] Vara J M, Bollati V A, Jiménez Á, et al. Dealing with traceability in the mdd of model transformations. IEEE Transactions on Software Engineering, 2014, 40(6): 555-583

[98] Alenazi M, Niu N, Savolainen J. A novel approach to tracing safety requirements and state-based design models//Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering. Seoul, South Korea, 2020: 848-860

[99] Eghan E E, Moslehi P, Rilling J, et al. The missing link-A semantic web based approach for integrating screencasts with security advisories. Information and Software Technology, 2020, 117: 1-16

[100] Linsbauer L, Angerer F, Grünbacher P, et al. Recovering feature-to-code mappings in mixed-variability software systems//Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution. Victoria, Canada, 2014: 426-430

[101] Ali N, Sharafl Z, Guéhéneuc Y G, et al. An empirical study on requirements traceability using eye-tracking//Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM). Trento, Italy, 2012:

191-200

[102] Ali N, Sharafi Z, Guéhéneuc Y G, et al. An empirical study on the importance of source code entities for requirements traceability. Empirical Software Engineering, 2015, 20(2): 442-478

[103] Walters B, Shaffer T, Sharif B, et al. Capturing software traceability links from developers' eye gazes//Proceedings of the 22nd International Conference on Program Comprehension. Hyderabad, India, 2014: 201-204

[104] Ahrens M. Towards automatic capturing of traceability links by combining eye tracking and interaction data//Proceedings of the 2020 IEEE 28th International Requirements Engineering Conference (RE). Zurich, Switzerland, 2020: 434-439

[105] Cleland-Huang J, Mäder P, Mirakhorli M, et al. Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders//Proceedings of the 2012 20th IEEE International Requirements Engineering Conference (RE). Chicago, USA, 2012: 231-240

[106] Mäder P, Gotel O. Towards automated traceability maintenance. Journal of Systems and Software, 2012, 85(10): 2205-2227

[107] Rahimi M, Cleland-Huang J. Evolving software trace links between requirements and source code. Empirical Software Engineering, 2018, 23(4): 2198-2231

[108] Ghabi A, Egyed A. Observations on the connectedness between requirements-to-code traces and calling relationships for trace validation//Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE). Lawrence, USA, 2011: 416-419

[109] Ghabi A, Egyed A. Code patterns for automatically validating requirements-to-code traces//Proceedings of the 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. Essen, Germany, 2012: 200-209

[110] Mills C, Escobar-Avila J, Haiduc S. Automatic traceability maintenance via machine learning classification//Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution. Madrid, Spain, 2018: 369-380

[111] Kuang H, Mäder P, Hu H, et al. Can method data dependencies support the assessment of traceability between requirements and source code? Journal of Software: Evolution and Process, 2015, 27(11): 838-866

[112] Charrada E B, Koziolek A, Glinz M. Identifying outdated requirements based on source code changes//Proceedings of the 2012 20th IEEE International Requirements Engineering Conference. Chicago, USA, 2012: 61-70

[113] Saini R, Mussbacher G, Guo J L C, et al. Towards queryable and traceable domain models//Proceedings of the 2020 IEEE 28th International Requirements Engineering Conference. Zurich, Switzerland, 2020: 334-339

[114] Niu N, Bhowmik T, Liu H, et al. Traceability-enabled refactoring for managing just-in-time requirements//Proceedings of 2014 IEEE 22nd International Requirements Engineering Conference. Karlskrona, Sweden, 2014: 133-142

[115] Niu N, Wang W, Gupta A. Gray links in the use of requirements traceability//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Seattle, USA, 2016: 384-395

[116] Krüger J, Çalıklı G, Berger T, et al. Effects of explicit feature traceability on program comprehension//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Tallinn, Estonia, 2019: 338-349

[117] Krüger J, Çalıklı G, Berger T, et al. How explicit feature traces did not impact developers' memory//Proceedings of 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering. Honolulu, USA, 2021: 610-613

[118] Dit B, Moritz E, Poshyvanyk D. A tracelab-based solution for creating, conducting, and sharing feature location experiments//Proceedings of the 2012 20th IEEE International Conference on Program Comprehension. Passau, Germany, 2012: 203-208

[119] Falessi D, Roll J, Guo J L C, et al. Leveraging historical associations between requirements and source code to identify impacted classes. IEEE Transactions on Software Engineering, 2018, 46(4): 420-441

[120] Mäder P, Egyed A. Do developers benefit from requirements traceability when evolving and maintaining a software system? Empirical Software Engineering, 2015, 20(2): 413-441

[121] Mirakhorli M, Cleland-Huang J. Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance//Proceedings of the 27th IEEE International Conference on Software Maintenance. Williamsburg, USA, 2011:123-132

[122] Dagenais B, Robillard M P. Using traceability links to recommend adaptive changes for documentation evolution. IEEE Transactions on Software Engineering, 2014, 40(11):1126-1146

[123] Agrawal A, Khoshmanesh S, Vierhauser M, et al. Leveraging artifact trees to evolve and reuse safety cases//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering. Montréal, Canada, 2019: 1222-1233

[124] Mäder P, Egyed A. Assessing the effect of requirements traceability for software maintenance//Proceedings of the 28th IEEE International Conference on Software Maintenance. Washington, USA, 2012:171-180

[125] Cetin H A. Identifying the most valuable developers using artifact traceability graphs//Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineer-

ing. Tallinn, Estonia, 2019:1196-1198

[126] Sülün E. Suggesting reviewers of software artifacts using traceability graphs//Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, USA, 2019:1250-1252

[127] Sülün E, Tüzün E, Doğrusöz U. RSTrace+: Reviewer suggestion using software artifact traceability graphs. Information and Software Technology, 2021, 130:1-13

[128] Ziftci C, Krüger I. Getting more from requirements traceability: requirements testing progress//Proceedings of the 7th International Workshop on Traceability in Emerging Forms of Software Engineering. San Francisco, USA, 2013:12-18

[129] Hayes J H, Dekhtyar A, Sundaram S K, et al. Requirements tracing on target (retro): Improving software maintenance through traceability recovery. Innovations in Systems and Software Engineering, 2007, 3(3): 193-202

[130] Lucia A, Oliveto R, Tortora G, et al. Adams re-trace: traceability link recovery via latent semantic indexing//Proceedings of the 30th International Conference on Software Engineering. Leipzig Germany, 2008:839-842

[131] Cleland-Huang J, Habrat R. Visual support in automated tracing//Proceedings of the 2nd International Workshop on Requirements Engineering Visualization. New Delhi, India, 2007:4-4

[132] Chen X, Hosking J, Grundy J. Visualizing traceability links between source code and documentation//Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing. Innsbruck, Austria, 2012:119-126

[133] Panichella A, McMillan C, Moritz E, et al. When and how using structural information to improve ir-based traceability recovery//Proceedings of the 17th European Conference on Software Maintenance and Reengineering. Genova, Italy, 2013:199-208

[134] Merten T, Jüppner D, Delater A. Improved representation of traceability links in requirements engineering knowledge using sunburst and netmap visualizations//Proceedings of the 4th International Workshop on Managing Requirements Knowledge. Trento, Italy, 2011:17-21

[135] Aung T W W, Huo H, Sui Y. Interactive traceability links visualization using hierarchical trace map//Proceedings of the IEEE International Conference on Software Maintenance and Evolution. Cleveland, USA, 2019:367-369

[136] Chen X, Hosking J, Grundy J, et al. DCTracVis: A system retrieving and visualizing traceability links between source code and documentation. Automated Software Engineering, 2018, 25(4): 703-741

[137] Lin Z, Marinov D, Zhong H, et al. JaConTeBe: A benchmark suite of real-world java concurrency bugs//Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. Lincoln, USA, 2015: 178-189

[138] Just R, Jalali D, Ernst M D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs//Proceedings of the 2014 International Symposium on Software Testing and Analysis. San Jose, USA, 2014:437-440

[139] Mahmoud A, Niu N. On the role of semantics in automated requirements tracing. Requirements Engineering, 2015, 20(3):281-300

[140] Vale T, de Almeida E S. Experimenting with information retrieval methods in the recovery of feature-code spl traces. Empirical Software Engineering, 2019, 24(3):1328-1368

[141] Qusef A, Bavota G, Oliveto R, et al. Evaluating test-to-code traceability recovery methods through controlled experiments. Journal of Software: Evolution and Process, 2013, 25(11):1167-1191

[142] White R, Krinke J, Tan R. Establishing multilevel test-to-code traceability links//Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering. Seoul, Korea, 2020:861-872

[143] Keenan E, Czauderna A, Leach G, et al. TraceLab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions//Proceedings of the 34th International Conference on Software Engineering. Zürich, Switzerland, 2012:1375-1378

[144] Zogaan W, Mujhid I, S Santos J C, et al. Automated training-set creation for software architecture traceability problem. Empirical Software Engineering, 2017, 22(3): 1028-1062

[145] Unterkalmsteiner M. Early requirements traceability with domain-specific taxonomies—A pilot experiment//Proceedings of the IEEE 28th International Requirements Engineering Conference. Zurich, Switzerland, 2020:322-327

[146] Maro S, Steghöfer J P. Capra: A configurable and extendable traceability management tool//Proceedings of the IEEE 24th International Requirements Engineering Conference. Beijing, China, 2016:407-408

[147] Lozano A, Noguera C, Jonckers V. Managing traceability links with MaTraca//Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering. Osaka, Japan, 2016:665-668

[148] Lin J, Liu Y, Cleland-Huang J. Supporting program comprehension through fast query response in large-scale systems//Proceedings of the 28th International Conference on Program Comprehension. Seoul, South Korea, 2020: 285-295

[149] Revelle M, Gethers M, Poshyvanyk D. Using structural and textual information to capture feature coupling in object-oriented software. Empirical Software Engineering, 2011, 16(6):773-811

[150] Qusef A, Bavota G, Oliveto R, et al. Scotch: Test-to-code traceability using slicing and conceptual coupling//Proceedings of the 27th IEEE International Conference on Software

Maintenance. Williamsburg, USA, 2011:63-72

[151] Pruski P, Lohar S, Aquanette R, et al. Tiqi: Towards natural language trace queries//Proceedings of the 2014 IEEE 22nd International Requirements Engineering Conference. Karlskrona, Sweden, 2014:123-132

[152] Burger A, Grüner S. FINALIsT^2: Feature identification, localization, and tracing tool//Proceedings of the IEEE 25th International Conference on Software Analysis, Evolution and Reengineering. Campobasso, Italy, 2018:532-537

[153] Ba,k K, Diskin Z, Antkiewicz M, et al. Clafer: Unifying class and feature modeling. Software & Systems Modeling, 2016, 15(3):811-845

[154] Bavota G, Colangelo L, De Lucia A, et al. TraCter: A tool for candidate traceability link clustering//Proceedings of the IEEE 19th International Requirements Engineering Conference. Trento, Italy, 2011:335-336

[155] Gall H, Hajek K, Jazayeri M. Detection of logical coupling based on product release history//Proceedings of the International Conference on Software Maintenance. Montpellier, France, 1998:190-198

WANG Ye, Ph. D., associate professor. Her current research interests include software engineering, machine learning and service computing.

HU Kun, master. His research interests include software engineering, machine learning and service computing.

JIANG Bo, Ph. D., professor. Her current research interests include software engineering, machine learning and service computing.

XIA Xin, Ph. D. His current research interests include data mining, software engineering and machine learning.

TANG Xian-Shu, undergraduate. Her current research interests include software engineering and machine learning.

Background

As an important software capability, software traceability aims to capture, link and trace each crucial software artifact and construct the traceability links between them. In concrete practice, software traceability is usually realized through traceability links. Traceability links refer to a specific relationship between a pair of software artifacts, one of which is the source artifact and the other is the target artifact. They records various dependencies, influences, causal relationships, etc. The direction of traceability links can be one-way or two-way. Most of the current research on software traceability focuses on traceability links, mainly because various traceability links can help software developers to understand, develop efficiently, and manage the system effectively. At the same time, traceability links can help people involved in all phases of software development activities to accomplish their development tasks.

A large number of researchers have proposed many techniques and approaches for the creation, maintenance, validation and application of traceability links. For example, natural language processing-based approaches, information retrieval-based approaches, machine learning-based approaches, deep learning-based approaches, etc. This paper is a systematic review on software traceability links approaches and techniques, which is mainly supported by Zhejiang Provincial Natural Science Foundation Project. The work of this paper benefits from the research experiences of the NSFC Project and Zhejiang Provincial Natural Science Foundation Project hosted by the author in recent years. All projects have focused on the following areas of research: intelligent software engineering and requirements engineering.

In this paper, we focus on automatic approaches to traceability links creation, maintenance, validation and application so as to sort out and summarize the research progress in the past ten years. The main content includes: 1) the statistics and analysis of approaches and techniques for creation, maintenance and validation of traceability links, 2) the application research of traceability links, 3) the state-of-the-art traceability links related tools, datasets and evaluation metrics, and 4) the key problems of the current traceability links techniques are summarized from the technical difficulties, and the possible solution ideas and future development trends of the problems are elaborated around seventh parts: the complexity of tracing software, the granularity problem, the unsatisfying accuracy, the type limitation, the validation efficiency, the application scale and time, and the incomprehensive evaluation of traceability links.

This paper summarizes several key issues and proposes solutions after reading and researching, which can help the reader to better understand the shortcomings and future trends in this field and generate thoughts accordingly.